AFRL-IF-RS-TR-2000-20
Final Technical Report
March 2000

DEVELOPMENT OF APPLICATION SOFTWARE HIERARCHY FOR REUSE (DASH'R)

Template Software, IBM, Honeywell, and ISX

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. C540

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-20 has been reviewed and is approved for publication.

APPROVED: Nancy A. Roberts

NANCY A. ROBERTS
Project Engineer

FOR THE DIRECTOR: Fowler

NORTHRUP FOWLER
Technical Advisor
Information Technology Division

DEVELOPMENT OF APPLICATION SOFTWARE HIERARCHY FOR REUSE (DASH'R)

Consortium of Companies & People

Contractor:	Template Software, IBM, Honeywell, and ISX
Contract Number:	F30602-95-2-0006
Effective Date of Contract:	08 April 1995
Contract Expiration Date:	30 June 1999
Program Code Number:	6.3.3
Short Title of Work:	Development of Application Software Hierarchy for Reuse (DASH'R)
Period of Work Covered:	Apr 95 – Jun 99
Principal Investigator:	Larry Sentman
Phone:	(703) 413-3106
AFRL Project Engineer:	Nancy Roberts
Phone:	(315) 330-3566

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Nancy Roberts, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
MARCH 2000

3. REPORT TYPE AND DATES COVERED
Final Apr 95 - Jun 99

4. TITLE AND SUBTITLE
DEVELOPMENT OF APPLICATION SOFTWARE HIERARCHY FOR REUSE (DASH'R)

5. FUNDING NUMBERS
C - F30602-95-2-0006
PE - 63570E
PR - C540
TA - 00
WU - 01

6. AUTHOR(S)
Consortium of Companies & People

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Template Software, Herndon VA
IBM, Austin TX
Honeywell, Minneapolis MN
ISX, Westlake Village CA

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTD
525 Brooks Road
13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-20

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Nancy Roberts/IFTD/(315) 330-3566

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The goal of this Technology Reinvestment Program Focus Area is to radically reduce the effort required to field new software applications through the development of reusable software components. Today an estimated 85% of the installed base is a custom application, with all components written especially for that software package. "Object Oriented Software," an emerging software technology, which is becoming widely used in the development of new software, offers the promise of reusability and ease of modification for both Defense and commercial applications. However, the promise can only be realized if the use of object oriented software is created according to an established set of standards and if appropriate reusable software components are developed. Building on emerging industry standards for software object technologies, these projects will significantly accelerate development of tools to help build the infrastructure for component ware, create a pool of developers experienced with applying the new tools, and deliver a series of demonstration applications with both commercial and defense relevance.

14. SUBJECT TERMS
Object Oriented Technology, Reuse, CORBA

15. NUMBER OF PAGES
134

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

EXECUTIVE SUMMARY: Technology Reinvestment Program

The Technology Reinvestment Program (TRP) is a multi-agency, dual-use technology investment effort that includes the Departments of Defense (represented by the Advanced Research Projects Agency and the three military Departments), Commerce, Energy, and Transportation, the National Science Foundation, and the National Aeronautics and Space Administration. It is the continuing mission of the TRP to simulate the transition to a growing, integrated, national industrial capability which provides the most advanced, affordable, military systems and the most competitive commercial products. TRP programs are structured to expand employment opportunities in dual-use United States industries and demonstrably enhance U.S. competitiveness and National security. The TRP fulfills its mission through the application of defense and commercial resources to develop dual-use technologies, to deploy manufacturing assistance to small firms, and to establish education and training programs that enhance U.S. manufacturing skills and target displaced defense industry workers. TRP was formed to execute statutory Programs enacted by Congress in the Defense Technology Conversion, Reinvestment, and Transition Act of 1992. These Programs have common requirements, including specified minimum participation with an emphasis on "partnerships", coat sharing between participants and the Federal Government, and defense relevance. All funds under these Programs are awarded through competitions.

The DASH'R (Development of Application Software Hierarchy for Reuse) consortium of Template Software, IBM, ISX, and Honeywell plans were to concurrently evolve and develop the following major technical elements as follows:

- A Core Technology Base (CTB) with integrated first generation CORBA 2.0 technology and complete with a high productivity visual software development and execution environment supporting Object Oriented development. This was accomplished by technology advancements of IBM and Template Software, Inc.
- A defense oriented Preventive Medicine Planning Tool (PMPT) a six phase interactive requirements and information process to assist Medical Planners in the development of a Preventive Medicine (PM) Plan, accomplished by ISX.
- A commercial in-line Batch Manufacturing Scheduling (BMS) application for orchestrating the completion of flexible manufacturing batches applicable to key U.S. industries, accomplished by Honeywell.

This report details the DASH'R TRP's success in completing these efforts.

Table Of Contents

List Of Figures

List Of Tables

Technology Reinvestment Program (TRP)

DASH'R (Development of Application Software Hierarchy for Reuse)

1. Introduction

The purpose of this project was to pursue research and development in "Object Technology for Rapid Software Development and Delivery". This project was authorized under AO#C540/PR C-5-2753. This project plan summarizes overall project coordination, elements, and methods.

1.1 Project Global Technical Objective

Principle Customer:

DARPA represented by Howard Shrobe DARPA / SISTO. Principal government contact for TRP completion was Nancy Roberts of Rome, NY.

- **Customer Vision** •[i]

The goal of this TRP Focus Area is to radically reduce the effort required to field new software applications through the development of reusable software components. Today an estimated 85% of the installed base of software is a custom application, with all components written especially for that software package. "Object Oriented Software," an emerging software technology, which is becoming widely used in the development of new software, offers the promise of reusability and ease of modification for both Defense and commercial applications. However, the promise can only be realized if the use of object oriented software is created according to an established set of standards and if appropriate reusable software components are developed. Building on emerging industry standards for software object technologies, these projects will significantly accelerate development of tools to help build the infrastructure for component ware, create a pool of developers experienced with applying the new tools, and deliver a series of demonstration applications with both commercial and defense relevance.

An increasing portion of DoD funding is directed to software development and maintenance. The Object Technology Focus Area will lead the way to radical reductions in these costs through the assembly of portable, interoperable software components into deployed applications. The three chosen projects will each provide for two prototype

[i] The Customer Vision is taken from the SOW and initial TRP documentation.

software developments, one for commercial and the other for military application. Perhaps as important, a collective impact of these efforts will be to promote shared, open, standards-based object technology, encouraging full access by the military to the commercial object-based software approach. If successful, this set of TRP activities will also avoid the compartmentalization of object-based software by strictly proprietary development.

Emerging object technologies and services promises to radically reduce the amount of new code required to field an application. Instead, new applications will be substantially assembled through the interconnection of application objects, object services, common facilities and an object request broker. The goal is to accelerate this technology's emergence. Projects will result in the creation of an object-technology-based design, development, and execution environment based on emerging industry standards. The environment should support a user-centered software life cycle model and multiple programming languages, be portable and interoperable with components and services from other providers in a distributed computing environment, and be potentially extensible to support real-time applications.

Projects are expected to result in functional demonstration applications, integrated design, development and run-time environments in the form of standards-based infrastructure, object services, common facilities, and application/ tool objects. Infrastructure and application development are expected to adhere to a user-centered process characterized by rapid system building, frequent incremental demonstrations, an aggressive schedule and strong team empowerment (including end-users and developers). Metrics for success include demonstrations of performance, cost to develop, utility (user's perspective, includes application developers and users), evolvability (ease of changing functionality, or add features), interoperability, and timeliness (speed of functionality delivery).

Project Details

The project is organized as a joint research and development Consortium to engage in a collaborative research effort of limited duration to gain further knowledge and understanding of technologies described in this plan. The consortium is founded on the Articles of collaboration for Object Technology for Rapid Software Development TRP Consortium dated TBD. DARPA is in partnership with the Consortium by way of a Cooperative Agreement Under 10 U.S.C. 2371 between The United States of America U.S. Air Force, Air Force Material Command Rome Lab and the consortium.

Additionally, there are two other consortiums in similar arrangements with DARPA to create "Object Technology for Rapid Software Development and Delivery". One team is comprised of Anderson consulting and their partners GoGenTex, Expersoft and Raytheon while the other is composed of I-Kinetics and their partners Heuristics Research, Iona Technologies, Navy/ Naval Sea Command, Netlinks Technology, SunSoft, and UTC-

Pratt & Whitney-Government Engines and Space Propulsion. The three consortiums will work together to demonstrate technology interoperability.

The project will be directed by a Consortium Management Committee (CMC) according to the articles of collaboration and managed according to **Figure 1 View of DASH'R Consortium and DARPA Management**.

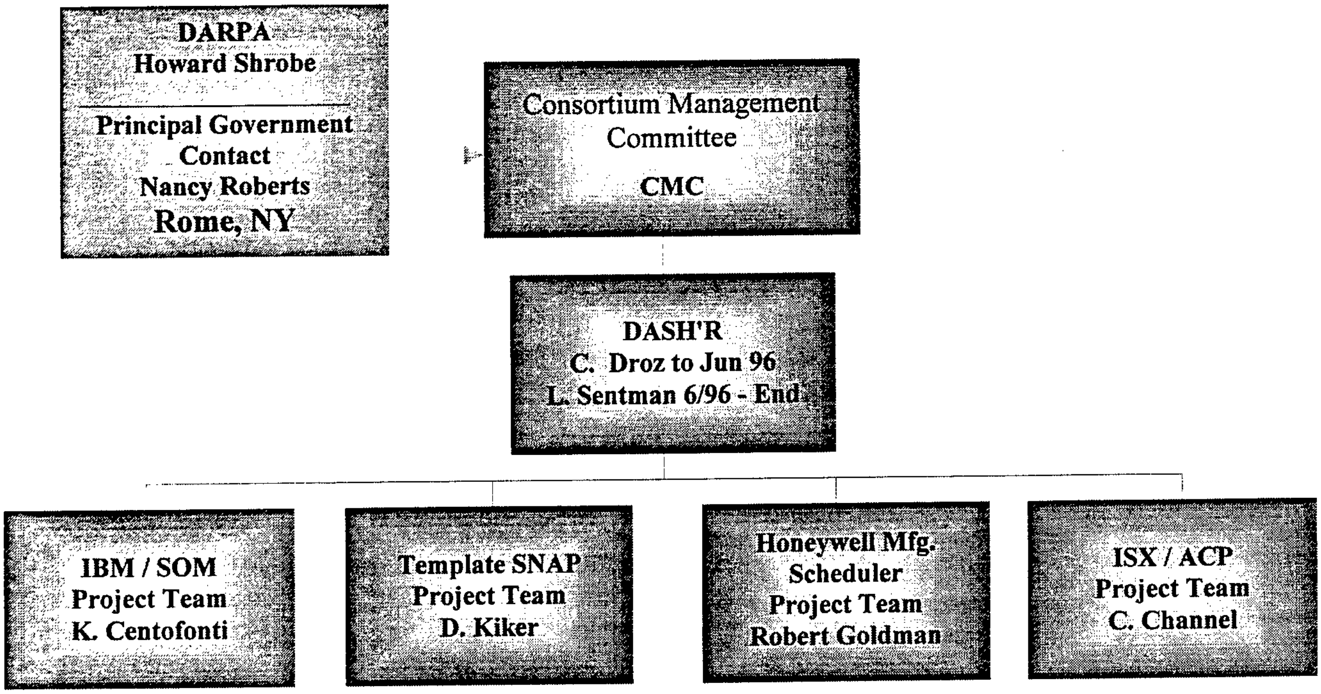


Figure 1 View of DASH'R Consortium and DARPA Management

Project Goals

The project effort will positively demonstrate next generation object technology capabilities for development and delivery of rapid software systems based on open standards. Core Technology Base combining CORBA based object technology and a comprehensive rapid software development environment provides the infrastructure for proposed efforts. Two planning and scheduling demonstration applications developed with core technology will show an existence proof of project goals over traditional practice.

Synopsis of Major Tasks and Responsibilities

The proposed effort will accelerate open, object technology infrastructure commercial availability. An end to end demonstration will provide an existence proof and will allow accumulation of experience with rapid software development and delivery. Finally, the project will conduct an interoperability test with other object technology consortia based on industry standard CORBA.

This plan schedules three major technology elements to support program objectives. An iterative approach and user-centered methods govern technical plan execution. The consortium will concurrently evolve and develop the major technical elements shown in **Table 1 DASH'R Consortium Tasks** below:

Table 1 DASH'R Consortium Tasks

Task(s)	***Contractor(s)***
A Core Technology Base (CTB) with integrated first generation CORBA technology and complete with a high productivity visual software development and execution environment supporting Object Oriented development.	IBM and Template
• CTB Visual Development Tools	Template
Campaign Planner Interoperability Demonstrations A defense oriented Preventive Medicine Planning Tool (PMPT) a six phase interactive requirements and information process to assist Medical Planners in the development of a Preventive Medicine (PM) Plan	ISX
A commercial in-line Batch Manufacturing Scheduling (BMS) application for orchestrating the completion of flexible manufacturing batches applicable to key U.S. industries.	Honeywell
Contract Coordination/ Administration	Template

Technology interaction is shown in **Figure 2 View of DASH'R Consortium and Technology Interaction** below.

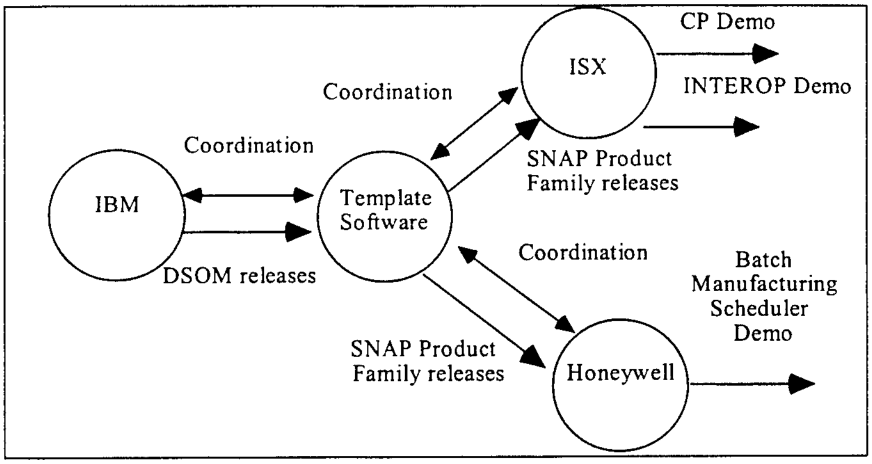


Figure 2 View of DASH'R Consortium and Technology Interaction

1.2 Implementation Strategy

Provision for management monitoring, control, and reporting are centered on an iterative, phased development approach. Phases will be generally be six to twelve months in duration. Each consortium member will have a phasing plan that will synchronize with others to support the "pipeline" dependencies suggested by the preceding diagram. The completion of phases serves not only to limit risk into finite R&D phases but also allows regular and frequent opportunity for DARPA and consortium management review of progress and level of effort. Given the research nature of the effort, the planning will also be updated on a phased basis by consortium members, one month before the phase begins. Appendix A contains the overall SOW and Appendix B the schedule.

1.3 Synopsis of Consortium Product Results

IBM provided SOMobjects development as a Core Technology. SOMobjects forms the basis for IBM's implementation of the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) and Object Services. SOM provides an object-structured protocol that allows applications to access and use objects, regardless of the programming language in which they were created and regardless of where the object resides on host/client or client/server networks. For enterprise-wide distributed computing, SOMobjects 3.0 brings critical flexibility to object-oriented programming, object reuse and sharing.

The SOMobjects 3.0 for AIX, OS/2 and Windows NT provides increased compliance with OMG's CORBA specifications and Object Services. SOMobjects 3.0 is commercially available via the Web.

Template Software's TRP effort was to provide an Interface to the IBM SOM/DSOM

Product Workflow Template (WFT) Visual Development Tools SNAP 8.0 language visual editors as a part of the Core Technology.

- • Interface to the IBM SOM/DSOM Product
  - SNAP 7.0 and the SNAP 7.6 products support a link to the SOMobjects 3.0.
  - Will continue support and update through the SNAP 8.0 product release.
- • Workflow Template (WFT) Visual Development Tools
  - WFT 3.6 was created to work with the SNAP 7.6 product.
    - WFT 3.6 has a series of enhancements in queue management that aid in very large-scale work flow systems
  - WFT 4.0 with improvements from WFT 3.6 is underway and will be real-time based
    - The WFT visual environment will be updated to take advantage of and be consistent with the new SNAP 8.0 visual development environment.

- SNAP 8.0 language visual editors was developed for release in First Quarter 1997
  - The baseline of SNAP 7.6 production environment is testing a wide range of graphic tools and graphic rewrites to support the new 8.0 editors
    - The underlying graphic system has been streamlined and modified to support geometry management. This SNAP Development Environment low level rewrite is complete and in SNAP 7.6 production version.
    - SNAP 7.6 has been ported and is in production in Windows
    - OS/2 porting is complete and will be released as a production system in SNAP 8.0
    - Implementation of new widgets (pane, drop down list, tabbed window, etc.) is complete and available in SNAP 7.6
    - See sample of SNAP 8.0 interface **Figure 3 SNAP Development Environment.**

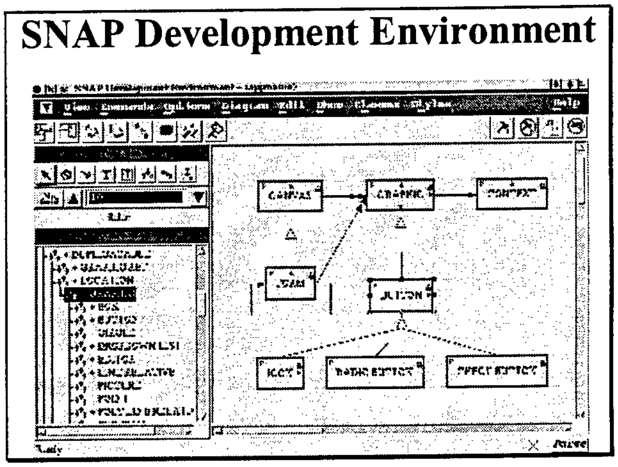


Figure 3 SNAP Development Environment

- The design of SNAP 8.0 was in Alpha release in mid-January 97
  - Visual editors are complete and implementation has begun.
  - SNAP C++ Generation A basic design for SNAP generation of C++ code is now in place.
  - See sample of SNAP 8.0 interface at the right.

Manufacturing Scheduling (BMS) application - ***Honeywell***:

- Spend rate smooth now that has departed from "front-loaded" plan.
- Experimenting with schedule server and UI client.
  - Developed broad outlines of client server architecture.
  - Developed IDL interface spec for architecture
  - Conducted preliminary experiments with architecture
- Publish CORBA interfaces.
  - Schedule Server, Comm. Object, View Object, Order Services, and Schedule Client.
- Completed development of core modules: search, Temporal Constraint Graph (TCG) and task and resource model.

Batch Manufacturing Scheduling (BMS) application - ***Honeywell***:

- Continue to develop the core technologies
  - Non-unary resources, e.g. storage tanks, work shifts
  - Better heuristics
- Work towards a demonstration system.

Campaign Planning Tool (CPT)/ Strategic Planning Toolkit (SPT) - ***ISX***:

- Strategic Planning Toolkit (SPT)
  - Glen Reece , Ph.D. From Edinburgh, Scotland, in charge of effort.
  - Technical Lead Assigned, meetings and design proceeding.
- Preventive Medicine Planning Tool (PMPT)
  - First release of PMPT to NEHIC made 10/24/96.
    - Next release week of 12/16/96.
  - Demonstration of PMPT to CHPPM conducted 10/24/96.
  - Ongoing ,meetings with NEHC and on PMPT. Discussions center on Industrial Hygiene, User Feedback, and Scenario possibilities.
  - Some implementation issues remaining with SOMobjects. Release in January of SOMobjects 3.0 will help solve the interface problems.
- Health Promotion Computer Associate (HPCA) with Copper's Institute and NEHC
  - Start date not set, waiting on NEHC funding.
- USE Documentation
- Rev. 1 released and being placed onto the WEB.
- Strategic Planning Toolkit
  - Meetings and design proceeding.
- Preventive Medicine Planning Tool (PMPT)

  - Ongoing ,meetings with NEHC and on PMPT. Discussions center on Industrial Hygiene, User Feedback, and Scenario possibilities.
- Planned PMPT showing at Conference in Virginia Beach, VA, at NEHC's booth in February 1997.
- Health Promotion Computer Associate (HPCA) with Copper's Institute and NEHC planned with no start date set.

DASH'R TRP
Final Report
Template Software, Inc.
April-95 through November-97

By: O. L. Sentman

2. Template Software, Inc.

Template Software, founded in 1978, is a publicly held company trading on the NASDAQ (TMPL). The company is headquartered in Dulles, Virginia, with European headquarters in Windsor, Berks, UK. The company has over 500 customers worldwide. Template has a growing multi-channel distribution network through direct sales, distributors and service partners that contribute to the rapid expansion of Template's presence worldwide.

The TRP was conducted during the time that Template Software became a public company. The support of DARPA in the TRP development was great aid in supporting Template's efforts to go public. The success of the Template products technically was helped greatly by our participation in the TRP.

2.1 Template Software Products

Overview

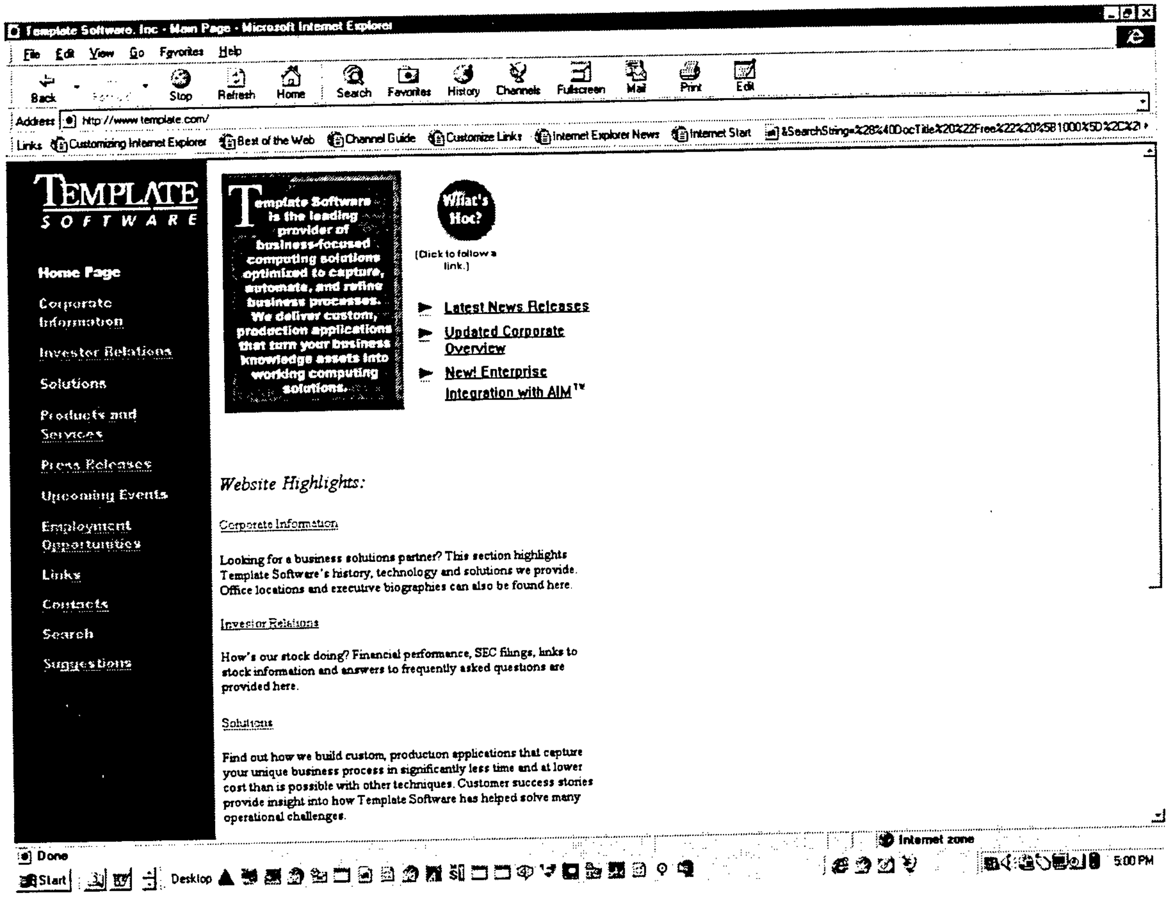
Offering both process automation and enterprise integration products, Template is dedicated to applying its advanced technology and expertise to meeting the challenges of the information enterprise. Template's advanced integration architecture and innovative business-oriented software templates enable the rapid delivery of custom solutions at package prices.

Process Automation

Template Software's enterprise-wide templates and turn-key business solutions combine the benefits of customized application solutions, with the cost and time-to-market benefits of off-the-shelf packaged applications. Our proven, reusable software templates allow each project to start with a baseline of functionality that can represent up to 90 percent of a complete, customizable application.

This pre-developed functionality, in the form of reusable objects, typically represents the portions of the application that are the most difficult to build and maintain and which do not add value through competitive differentiation. Free to focus on the business problem, Template is able to deliver high-quality, high-value process automation solutions in less time and at lower cost.

For more extensive information on the current extent and capabilities of Template Software's products, visit Template's WEB site (See **Figure 4 Template Software WEB Home Page** below) located at http://www.template.com.



Template Software is the leading provider of business-focused computing solutions optimized to capture, automate, and refine business processes. We deliver custom, production applications that turn your business knowledge assets into working computing solutions.

- Latest News Releases
- Updated Corporate Overview
- New! Enterprise Integration with AIM™

Website Highlights:

Corporate Information

Looking for a business solutions partner? This section highlights Template Software's history, technology and solutions we provide. Office locations and executive biographies can also be found here.

Investor Relations

How's our stock doing? Financial performance, SEC filings, links to stock information and answers to frequently asked questions are provided here.

Solutions

Find out how we build custom, production applications that capture your unique business process in significantly less time and at lower cost than is possible with other techniques. Customer success stories provide insight into how Template Software has helped solve many operational challenges.

Figure 4 Template Software WEB Home Page

Market View

Shrinking global markets and deregulation mean that more companies are competing for the same market segment. Differentiation is required, time-to-market is increasingly critical and cost containment vital. "Getting there first" is more important than ever, placing greater pressure on rapid development and deployment of business-critical applications.

To compete effectively in this dynamic environment, organizations have shifted focus from "what" products they make to "how" they make and deliver them. Applications that improve an organization's essential business processes – and add competitive Template Software's distributed object technology, uniquely packaged into reusable Software Templates, enables a new model-driven approach to software development, allowing organizations to innovate while preserving legacy investments. Capitalizing on this new approach, we deliver enterprise-wide solutions in significantly less time and at a lower cost than is possible with other techniques. We are uniquely positioned to address your most demanding business computing requirements - turning your business knowledge assets into working computing solutions.

The following diagram (**Figure 5 Hierarchy of Templates**) illustrates our hierarchy of templates and how they are designed to be used together to provide as much pre-written software code as possible in developing customized applications:

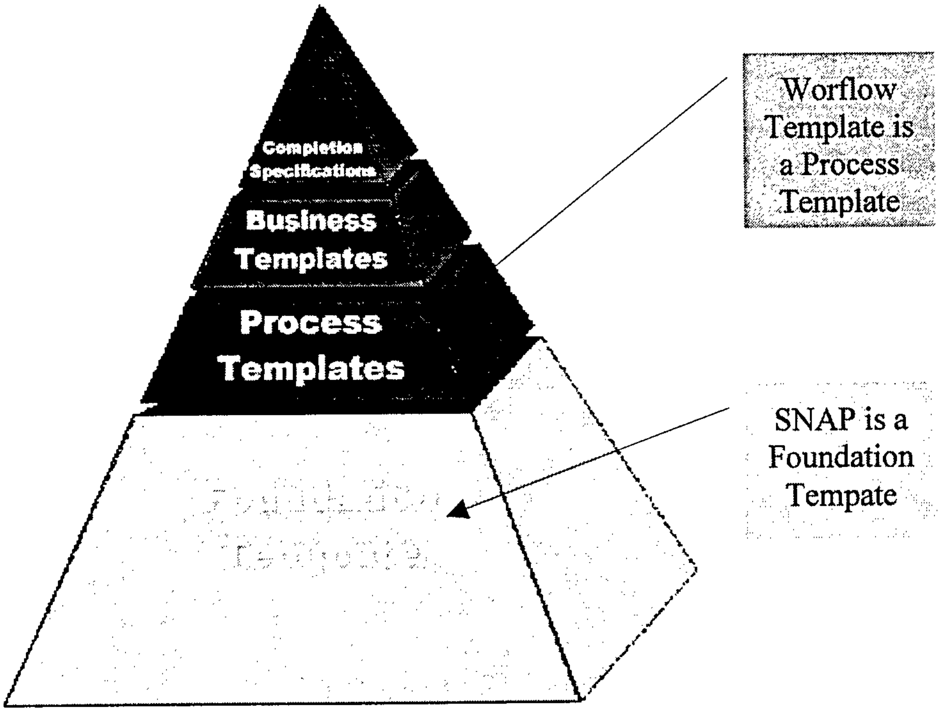

Completion Specifications
Business Templates
Process Templates
Foundation Template
Worflow Template is a Process Template
SNAP is a Foundation Tempate

Figure 5 Hierarchy of Templates

2.2 SNAP and Workflow Within the TRP - Overview

The Template Software objective was to create enhancements to its SNAP Product Family supporting consortium goals. The SNAP Product Family, as described above, is object technology-based development environments for peer-to-peer distributed applications. Planned enhancements fall into three categories — SOM integration, C++ code generation, and visual tools development. SOM integration will provide a CORBA standards-based substrate to the SNAP Product Family for object interoperability and services. C++ code generation eliminates applications dependence on Template development tools for maintenance, in step with TRP standards thrust and in the direction of interoperability at the development environment level. Incorporation of visual development tools will enhance SNAP Product Family's characteristics of rapid application and user-centered development.

A.3.1 Work Effort Partitioning

SOM integration extensions to the SNAP Product Family are independent from visual tool enhancement and define a natural work effort partitioning.

SOM Integration — the primary objective of SOM integration is to make the SNAP Product Family both a producer and consumer of reusable object services through SOM. To accomplish this, the External Application Software Component of SNAP will be extended to integrate SOM at the emitter framework level. This level of integration will make it possible for a SNAP developer to use externally created object components simply by writing method invocations in the SNAP language. Conversely, SNAP objects will be accessible to non-SNAP programmers writing in other SOM-compliant languages.

As ISX and Honeywell experience grows with this capability, other extensions regarding SOM may be made. Such extensions might include integration of selected SOM services and better leveraging of SNAP Product Family object services via SOM

C++ code generation — the SNAP Product Family Development Environments will be enhanced to allow generation of C++ class hierarchies

Visual Tools Enhancement — the SNAP Product Family Development Environments will be enhanced with new visual tools to simplify and speed up the development process. Anticipated extensions include:

- A new visual editor to graphically specify an application process object model.
- A language editor that can be invoked as needed by the various visual editors.
- Visual editors allowing end to end development, where end to end development includes business process analysis through system deployment
- Support for team development of multiprocess applications.

2.3 SNAP and Workflow Within the TRP - Results

Template Software's TRP effort was to provide an Interface to the IBM SOM/DSOM Product Workflow Template (WFT) Visual Development Tools SNAP 8.0 language visual editors as a part of the Core Technology. The following are the results of that effort:

- Interface to the IBM SOM/DSOM Product
  - SNAP 7.0 and the SNAP 7.6 products support a link to the SOMobjects 3.0.
  - Will continue support and update through the SNAP 8.0 product release.

- Workflow Template (WFT) Visual Development Tools
  - WFT 3.6 was created to work with the SNAP 7.6 product.
  - WFT 3.6 has a series of enhancements in queue management that aid in very large scale work flow systems
  - WFT 4.0 with improvements from WFT 3.6 is underway and will be run-time based
  - The WFT visual environment will be updated to take advantage of and be consistent with the new SNAP 8.0 visual development environment.

- SNAP 8.0 language visual editors was developed for release in First Quarter 1997
  - The baseline of SNAP 7.6 production environment is testing a wide range of graphic tools and graphic rewrites to support the new 8.0 editors
  - The underlying graphic system has been streamlined and modified to support geometry management. This SNAP Development Environment low level rewrite is complete and in SNAP 7.6 production version.
  - SNAP 7.6 has been ported and is in production in Windows
  - OS/2 porting is complete and was released as a production system in SNAP 8.0
  - Implementation of new widgets (pane, drop down list, tabbed window, etc.) is complete and available in SNAP 7.6
  - See sample of SNAP 8.0 interface at the right **(Figure 6 SNAP Development Environment)**.

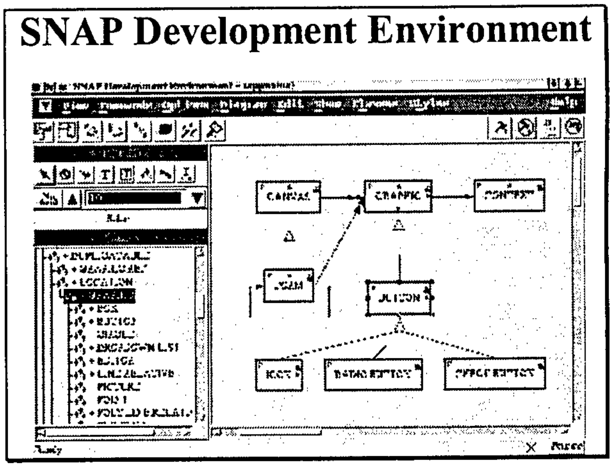


Figure 6 SNAP Development Environment

- The design of SNAP 8.0 was in Alpha release in mid-January 97
  - Visual editors are complete and implementation has begun.
  - SNAP C++ Generation A basic design for SNAP generation of C++ code is now in place.
  - See sample of SNAP 8.0 interface at the right.

2.4 Screen Shots of the Current State of SNAP and Workflow Development

The screen shots in this appendix show the current state of the Workflow and SNAP interfaces.

The SNAP Development Environment main window appears when you start the SNAP Development Environment. Examples of the windows in the SNAP Development Environment are shown in **Figure 7 The WorkFlow Main Window, Figure 8 The WorkFlow Task Editor, Figure 9 The SNAP Development Environment**, and **Figure 10 The SNAP Development Environment**. Note that the application name appears in the title bar of the window.

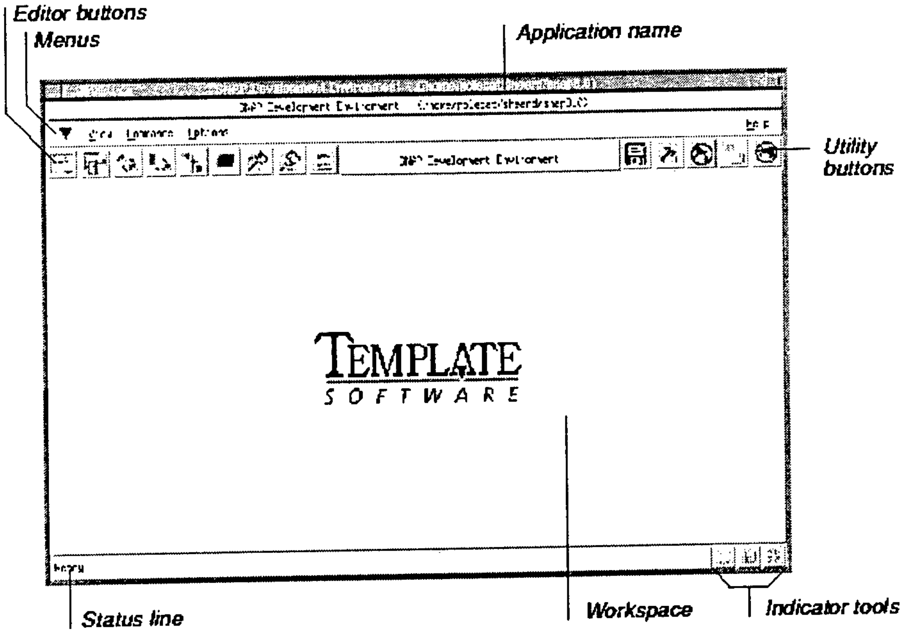


Figure 7 The WorkFlow Main Window

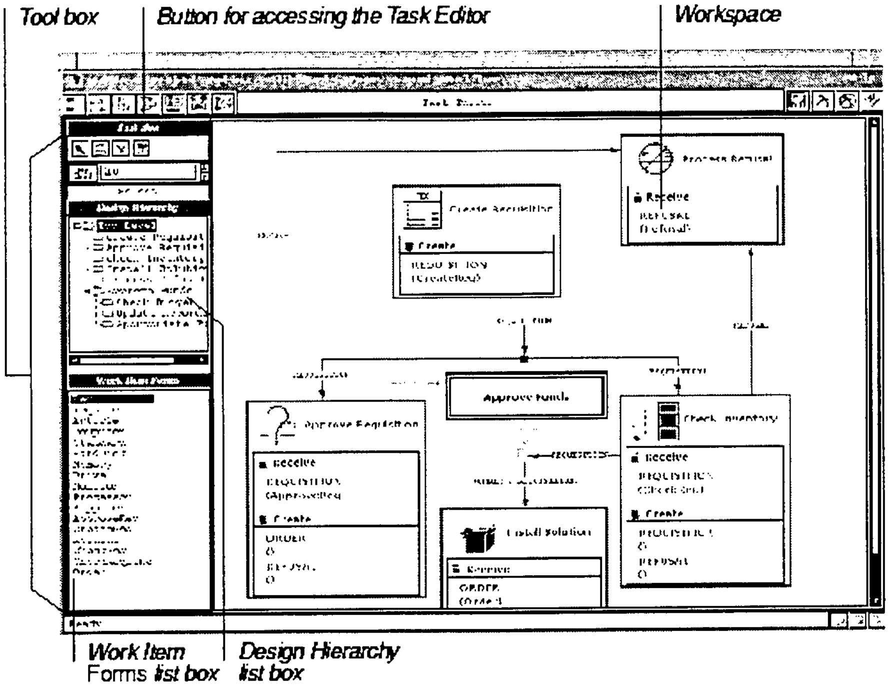
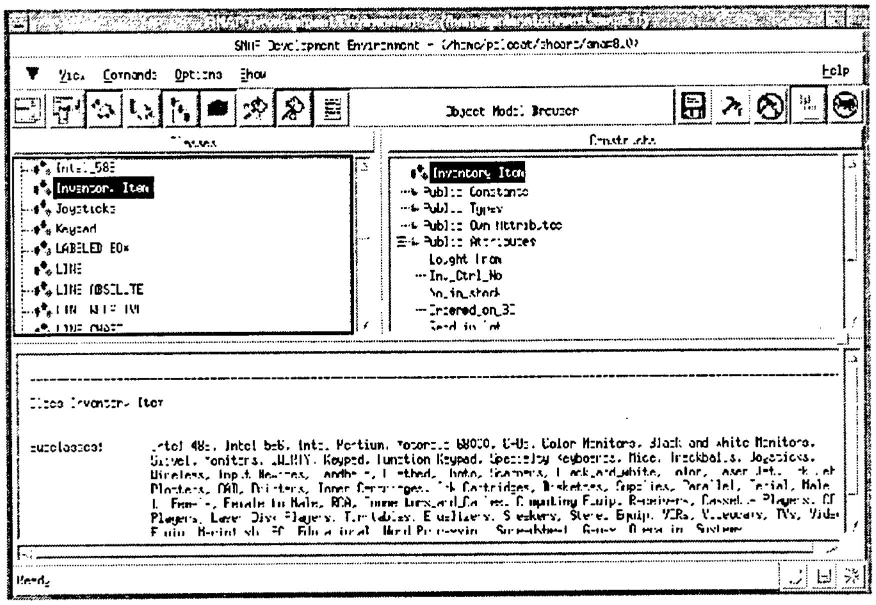


Figure 8 The WorkFlow Task Editor



Class Inventory Item

subclasses: Intel 486, Intel 586, Intel Pentium, Motorola 68000, CPUs, Color Monitors, Black and White Monitors, Swivel Monitors, QWERTY Keypad, Function Keypad, Specialty Keyboards, Mice, Trackballs, Joysticks, Wireless, Input Devices, Handhelds, Flatbed, Photo Scanners, Black_and_white, Color, Laser Jet, Ink Jet, Plotters, CAD Printers, Toner Cartridges, Ink Cartridges, Diskettes, Supplies, Parallel, Serial, Male to Female, Female to Male, RCA, Connectors_and_Cables, Computing Equip, Receivers, Cassette Players, CD Players, Laser Disc Players, Turntables, Equalizers, Speakers, Stereo Equip, VCRs, Camcorders, TVs, Video Equip, Macintosh, PC, Educational, Word Processing, Spreadsheet, Games, Operating Systems

Figure 9 The SNAP Development Environment

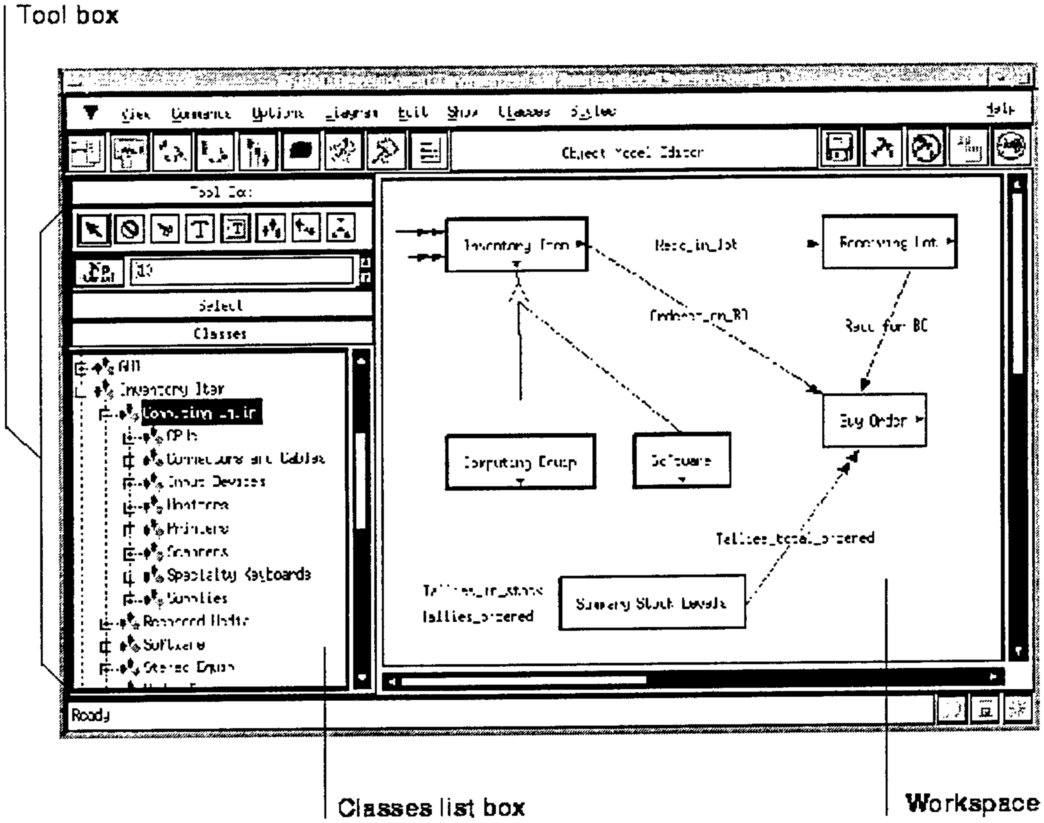


Figure 10 The SNAP Development Environment

2.5 SNAP and Workflow Within the TRP - Results

ISX noted some issues that pointed towards advantages to SNAP and also some disadvantages. Those comments are listed here. Improvements to SNAP 8.0 have corrected all problems noted by ISX. The following table, **Table 2 ISX Reported SNAP Advantages and Disadvantages,** indicates that status.

Table 2 ISX Reported SNAP Advantages and Disadvantages

PROBLEM	***FIXED IN SNAP 8.0***
Advantages of SNAP	
1. Has allowed for the rapid creation of a prototype system.	**NA**
2. SNAP includes ways to indirectly reference and manipulate classes, objects and attributes that has proved useful.	**NA**

3. A "tab card" widget for modifying objects was provided by SNAP and has proved useful. This widget is still experimental and needs refinement.	**NA**
4. Code developed using SNAP is cross-platform and can be easily ported to other operating systems. We will experiment with this feature by porting the PMPT to SNAP running under Windows NT.	**NA**
Problems with SNAP	
1. Custom relational attribute modification dialogs are better supported in 7.0, but are still cumbersome.	**YES**
2. No built in hypertext help supported (plans for HTML support are being considered)	**YES**
3. No support for "wave over" or help text or "tool tips" (planned for SNAP 7.5)	**YES**
4. Spin control and pull down lists do not function or are not available (not planned)	**YES**
5. Some of the development features found in commercial PC software development systems that are inadequate in the SNAP development environment include: a. The debugger is cumbersome and inadequate for easily visualizing program execution (scheduled to be improved in SNAP 8.0) b. A separate debugger must be used to visualize C code c. A secondary editor is required to write code, the integrated code editor is inadequate (scheduled to be improved in SNAP 8.0) d. Graphic objects can not be copied and pasted between two canvases (Change request submitted) e. No online help/examples are available, only online reference matl.	**YES**

DASH'R TRP
Final Report
ISX
April-95 through November-97

By: Charles L. Channell

3. ISX

ISX today is in the business of providing Advanced Concept Engineering (ACE) services. We engineer systems that leverage advanced technologies to solve operational problems, and programs that are designed to facilitate rapid development and maturation of new technology.

ISX applies innovative engineering methods, known as User-centered Software Engineering (USE), to rapidly design, prototype, and deliver computer system solutions that solve users' real-world operational problems. ISX engineers are exceptionally skilled at working with user/experts to quickly grasp operational problems, to visualize feasible automation solutions, and to communicate that vision both to decision-makers and to the software implementation. The development team uses scenario-based development, rapid prototyping, and continuous intensive involvement by users in all phases of the system design and development. ISX has successfully applied these methods to operational problems in many domains.

3.1 Overview

This is ISX's Final Report on ISX's effort under the Dynamic Analysis of Software Hierarchy for Reuse Technology Reinvestment Program (DASH'R TRP) originally known as the Object Technology for Rapid Software Development (OTRSD) TRP. This effort started in April 1995 and was to last 2 years through April 1997, but was extended through November 30, 1997 as a no cost extension.

For this TRP, ISX was a member of a consortium of companies which consisted of Template Software as the prime contractor, with IBM, Honeywell and ISX. The overall intent of the TRP was to produce object oriented software development environments/tools that could be effectively used in both the commercial and military domains. IBM and Template produced the Core Technology Base (CTB) and Honeywell and ISX produced proof of concept applications that employed the CTB (see **Figure 11 Organization Product Relationships**). Honeywell operated in the commercial domain and ISX in the military domain. This report focuses upon ISX's effort.

ISX used the CTB for two applications. One application, the Preventive Medicine Planning Tool (PMPT), was defined from the beginning of the program and helped to drive the requirements fed back to the technology providers on how appropriate the CTB was to military applications. The second application, a Navy Medical Waiver

Administration Tool, was started towards the end of the TRP period of performance and validated the improved CTB.

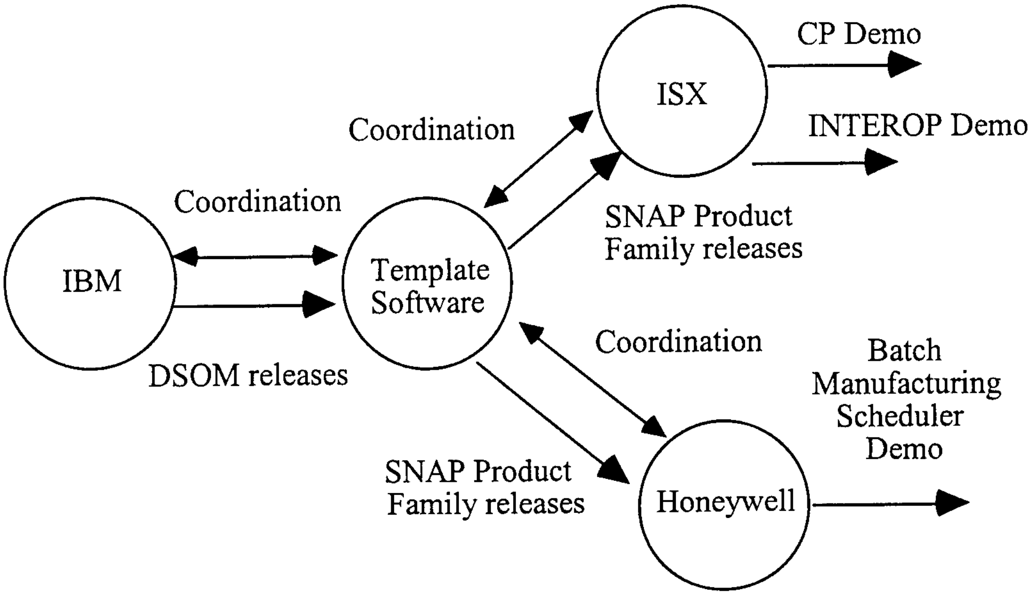


Figure 11 Organization Product Relationships

3.2 Background

ISX had the following tasks:

- Documenting and distributing the Used Centered Software Development (UCSD) methodology to the consortium members;
- Creating the Strategic Planning Toolkit (SPT) which is a generic version of the Air Campaign Planning Toolkit (ACPT);
- Development of a military application using the CTB and based upon ISX's Air Campaign Planning Toolkit (ACPT).

3.2.1 UCSD

One of the first tasks that ISX started under the TRP was the UCSD effort. This task consisted of surveying projects throughout ISX for the extent that the UCSD approach was used. The best examples were then explored in greater detail and put into the UCSD document.

3.2.2 SPT

Another product of the TRP, was a generic strategic planning toolkit which employed the ACPT's strategy-to-task (STT) planning methodology. ISX funded a multi-year effort using ISX's TRP cash contribution in the form of IRAD funds. The SPT was driven by the requirements from various programs including air campaign planning and preventive medicine deployment planning.

3.3 Military Application

The intent of the military application effort was to provide a "proof of applicability" of the CTB to military applications and to provide an example set of requirements to the CTB development effort. The original choice of the military application was a rehost of the ACPT into the CTB. However, it was decided to not be worthwhile as the rehost approach wouldn't deliver to the end-users anything that they didn't already have along with the added burden of creating two separately maintained ACPTs. Further, the CTB version would be at least 1 year behind the original version. As a result, the decision was made to select a new domain and a new application. ISX was given latitude regarding the domain selection for the military application with the major criteria of "the end users must like the end product and applaud the development process". Eventually, it was decided to work with the Navy Environmental Health Center (NEHC (pronounced "knee-hick")) to create a Preventive Medicine Planning Tool.

Towards the end of the project, about $70k of additional funds were made available to ISX with the consortium deciding to pursue a different domain and to employ as much of the CTB as possible. From working with CAPT Breeden (now of the Navy's Bureau of Medicine and Surgery (BuMed)) on the PMPT, he pointed the team to CAPT Jones, also of BuMed, and the domain of Navy Medical Waivers. The final project was decided to see how far ISX could get in putting together a Medical Waiver Application tool.

3.3.1 ACE/USE

This section encompasses the brief history of ACE components, accompanying terms used in the ACE methodology, and the results of a survey of ISX employees during the summer of 1995 under the TRP effort. Advanced Concept Engineering (ACE) is the methodology that enables ISX to continue to develop software systems that exceed customer and user expectations, while reaching new capabilities with every project.

ISX embraced software industry concepts of rapid prototyping and incremental delivery soon after its formation; and modified the standard Waterfall Methodology to avoid dependence on overly precise requirements that hampered development. The modified Waterfall was called Intelligent Systems Engineering (ISE), and was followed during the development of the Pilot's Associate. ISX soon realized that ISE set up conditions which forced freezing requirements for long periods of time. Ultimately, engineers concluded that without direct user interaction during the early development period, the delivered results were below expectations.

Steps were taken to improve user interaction during initial prototype development and maintained through full development in programs such as the Air Campaign Planning Tool. This strategy became a guide as engineers quickly realized the value of user input beyond initial requirements. User-Centric Software Engineering (USE), also called User Centered Software Development (UCSD), refined rapid prototyping, eliminated middlemen, and switched system development focus from solving an operational problem to creating greater capabilities for the user.

Key ISX engineers came to the realization that—to more fully enable transfer of new technologies into the mainstream— one must manage and encourage the development of new technologies ***and*** manage the environments (programs) in which these technologies can first come to bear, ***as well as*** the organizations controlling these environments. ACE integrates USE and Advanced Technology Management (ATM - evaluates emerging technologies to best fit the user's needs) to manage organizations, programs, or systems. ACE combines ATM principles with more efficient management of multi-use projects. In the future, ACE will expand to include many other methods, tools, and concepts.

Development trends in the software industry are coming to a crossroads between open and closed architectures. Both schools of thought have their proponents and detractors, however ISX is currently not in the position to determine how commercial software will be designed. Understanding this, it has been to ISX's advantage to follow the ACE methodology to determine the most appropriate architecture for a task. As ISX's position in the industry grows stronger, it will have more control over the commercial architectures it chooses to integrate and interoperate with. While ISX is evolving so will ACE. Current constraints will diminish as ISX, through ACE, becomes more proficient at managing its technologies, growing core competencies, and drawing more input from users.

3.3.2 PMPT

3.3.3 Domain Description

Preventive Medicine (PM) deals with preventing the military fighting units from suffering casualties due to disease and other non-battle injuries (DNBI). Usually, this entails educating the soldiers about the medical and environmental threats, providing prophylaxis and ensuring that proper sanitary procedures are followed. PM in the deployment context used in this effort deals with populations and not with individuals. Medical threats include diseases (e.g. malaria) and the disease vectors (e.g. mosquitoes). Environmental threats could include contaminated soils (e.g. heavy metal pollution of the top soils and water supplies), water and air in a general area or a specific area (e.g. a destroyed manufacturing area). Another threat is that posed by nuclear, biological or chemical (NBC) weapons and the manufacture, storage and deployment of those weapons.

When a deployment of US soldiers is planned, an OPLAN is created which outlines the mission, the chain of command, what units are participating, where they are going, etc. One component of the OPLAN is the Medical appendix of which the Preventive Medicine subcomponent is an annex. In the planning process, there is usually one medical officer assigned whose specialty within the medical domain varies. Due to the nature of preventive medicine, a large portion of the PM activities need to occur at the very beginning of the planning process. These activities can include determining which shots need to be administered to the deploying troops and then having those shots given. To be effective, the shots need to be given some time before exposure to the medical threats occur.

The primary end-user for the PMPT is the preventive medical officer (PMO) assigned to the Joint Task Force (JTF) Surgeon's Staff. This PMO could be inexperienced in deployment planning or could be a medical officer without PM training. An inexperienced PMO is a possibility because the US military usually rotates its personnel every 2-4 years. Hence, just as a person gains a good level of experience in a particular position, they are rotated out and a new, inexperienced person takes over. A MO without PM training is a possibility because a PMO wasn't handy at the time that assignments were being made.

These aspects of the PM domain and the military planning process allow for automation via the PMPT to improve the overall process. Some of the PMPT features that improve the PM planning process are:

- Mistake Avoidance: The PMPT will step a neophyte Preventive Medicine Officer (PMO) or a medical officer without PM training, through the PM planning process. By going through the process, the PMO will be exposed to the major issues of PM and will have to address them. Thus, avoiding overlooking any major "gotchas".

- One-stop Information: A PMO uses a small set of information sources to do the majority of their planning. These sources include the US Army's MEDIC, the US Navy's DISREP and VECTRAP, the CIA's World Fact Book, the CCDM manual, etc. Most of these sources are available electronically and can be accessed with the PMPT.

- Provide Guidance in creation of PM portion of OPLAN: In stepping the PMO through the PM planning process, much of the information about the mission and the medical threats have been entered into the computer. This information is then used by the PMPT to provide rough draft versions of the PM portion of the OPLAN, emphasis letters to the chain of command, to-do lists for the PMO, etc. These documents are available electronically and can then be edited further by the PMO to produce the final versions.

The PMO will be following three major steps (See **Figure 12 Preventive Medicine Planning Process**):

- Determine Requirements: What are the requirements of the troops/mission and what medical threats will need to addressed.

- Situation Definition: Given the medical threats and what resources a PMO can expect, what additional resources will be required to adequately address the medical threats.

- Generate Outputs: Once all the information has been entered, the PMPT can generate draft versions of the documents needed by the PMO.

The first set of information that is needed is the mission statement and the definition of the mission. This includes what troops are involved, why they are involved, when and for how long, what type of mission will it be (e.g. combat, keeping the peace, etc.), etc. In the current version of the PMPT, this information is entered by the PMO. In later versions of the PMPT, this information could be downloaded from an existing source. From this information, the populations involved can be derived along with locations. This includes how many US troops will be involved, their origination point, their destination, any intermediate points, other populations involved (e.g. refugees, NATO soldiers) and where they came from, are going to and through. Thus, the PMPT is creating lists describing the populations and countries/regions of interest to the PMO.

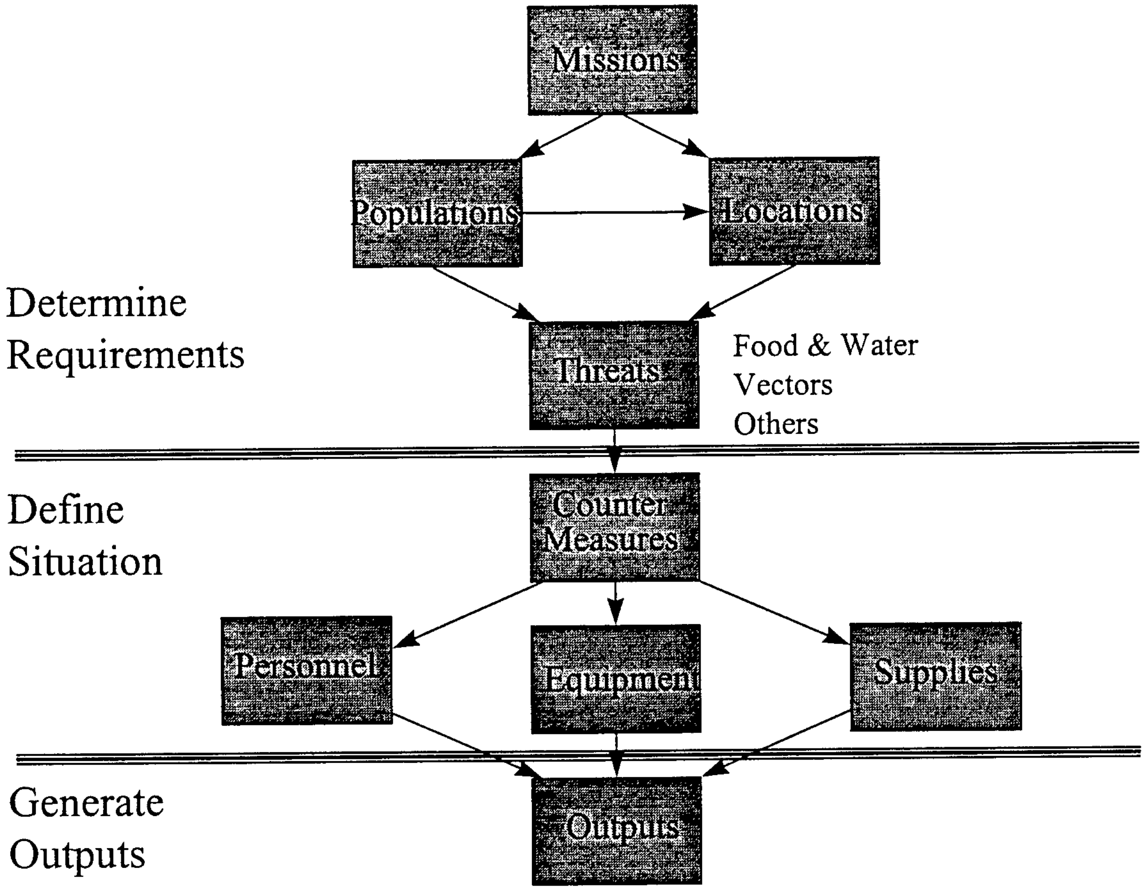


Figure 12 Preventive Medicine Planning Process

Based upon the populations and locations lists, the PMO can now refer to the standard information sources to determine the likely medical threats. The locations information coupled with the time of year (e.g. rainy season) will dictate much about the environmental threats and the diseases and vectors that will be encountered by the US personnel. The populations information dictates how sanitary are the practices of the populations, any social customs that need to be taken into account (e.g. separation of male and female adult populations), population demographics and total quantities of populations.

Based upon these lists of medical threats, the PMO uses his judgment to prioritize the lists and then uses the lists to set up a counter-measures list. The counter-measures includes the entire range of activities from sending out orientation booklets to the populations, to providing inoculations, to spraying for vectors, to monitoring the incidence of DNBI, to providing sanitary facilities to refugee populations, etc. All of these countermeasures require a certain amount of personnel, equipment and/or supplies to properly fulfill and will be based upon the total number of populations involved and the physical distribution of those populations. Thus, the next step of the PMPT is generate three more lists of the equipment needed along with personnel and supplies to address the countermeasures. Some of these materials/personnel can be provided organically by the units participating in the deployment. The PMO can now examine the lists and determine what is still needed and can then work to find out how these materials/personnel can be supplied.

Since the PMPT now contains a large set of information about the mission, it can now generate outputs automatically which the PMO can then edit. Within the PMPT are various common formats used in previous deployments and are available for the PMO to chose between to get a strong start.

Additional advantages provided by the PMPT include its ability to be used by non-PM trained personnel and in its bundling of various PM information materials. This can then be used as the basis for general information about regions.

A future version of the PMPT could be tied in with the electronic OPLAN and could upload the latest PM annex and could download the latest mission plan etc. and then be used to modify the PM annex.

During execution, a future version of the PMPT could be used to monitor the status of the PM plan. It could provide standard forms (e.g. DNBI incidence reports). Accumulate information and provide part of the lessons learned basis.

3.3.4 Healthcare & Information Technologies Group

One of the major applications that ISX worked on under the TRP, was a Preventive Medicine Planning Tool. The Navy Environmental Health Center provided domain expertise along with access to domain experts from the various services, to help define and refine a tool that assists the CING Surgeon and his staff in defining the Preventive Medicine portion of the OPLAN. Some of the tool highlights are:

- Preventive Medicine Planning Tool (PMPT): Identifies impacts of plan changes and provides guidance to Preventive Medicine Professionals in developing a preventive medicine plan;
- Intelligent Information Integration: Provides easy access for software to information stored in multiple heterogeneous data sources;
- Medical Readiness Strategic Plan Evaluation Tool (MiRSP-ET): Assists the Department of Defense (Health Affairs) personnel with authoring and evaluation of the MRSP;
- Information Support for Law Enforcement: Assists law enforcement agents with case development, management, and analysis;
- Environment Restoration Information System II: Assists a program manager in charge of a restoration site to manage the data for a project.

Currently, ISX is working with NEHC and the Navy Bureau of Medicine and Surgery to further develop the tool and deploy it across the services.

Customers for these projects include the Defense Advanced Research Projects Agency, the Idaho National Engineering Laboratory, the FBI and the Office of the Assistant Secretary of Defense (Health Affairs). The HIT personnel are skilled in working on a variety of platforms using the latest development tools and object oriented programming techniques.

The Preventive Medicine Planning Tool (PMPT) has been developed as one of the programs of the TRP called Object Technology for Rapid Software Development. The team consists of the Defense Advanced Research Projects Agency, the Honeywell Technology Center, IBM, ISX Corporation, and Template Software. The goal of the project is to utilize a core technology base to demonstrate how such technology can (1) improve developer productivity, (2) reduce time-to-market, and (3) facilitate the reuse of system components.

3.3.5 Visionary Demonstration

The first step ISX took in creating the PMPT was to work with the NEHC personnel to determine the overall requirements of the tool. This process employed ISX's Advanced Concept Engineering (ACE) process to capture the requirements and to provide a mechanism for the end-users to provide feedback to the tool designers. A key part of this process was the creation of a Visionary Demonstration (viz-dem) which is a non-functioning prototype that captures the look and feel of the final prototype quickly. (See **Figure 13 PMPT Visionary Demonstration Introduction Screen, Figure 14 PMPT Visionary Demonstration Overview Screen, Figure 15 PMPT Visionary Demonstration SPT Architecture Screen**, and **Figure 16 PMPT Visionary Demonstration Mission Query Screen**). This then allows the developers to meet with the end-users, extract information about the PM process, codify the process in the viz-dem, demonstrate the proposed process to the end-users, and then receive feedback from the end-users. This provides a sanity check to the whole process without a significant investment in time or resources before a discontinuity is detected between the developers and the end-users.

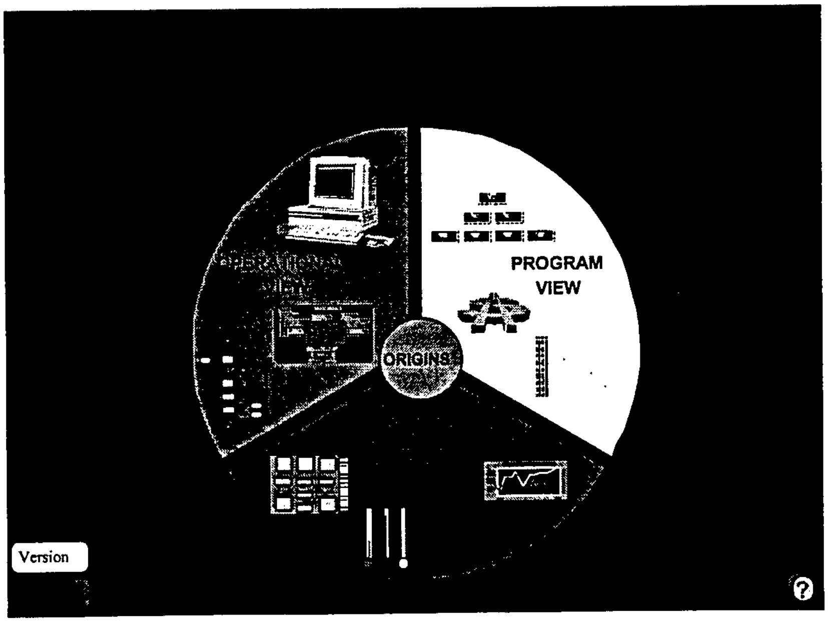
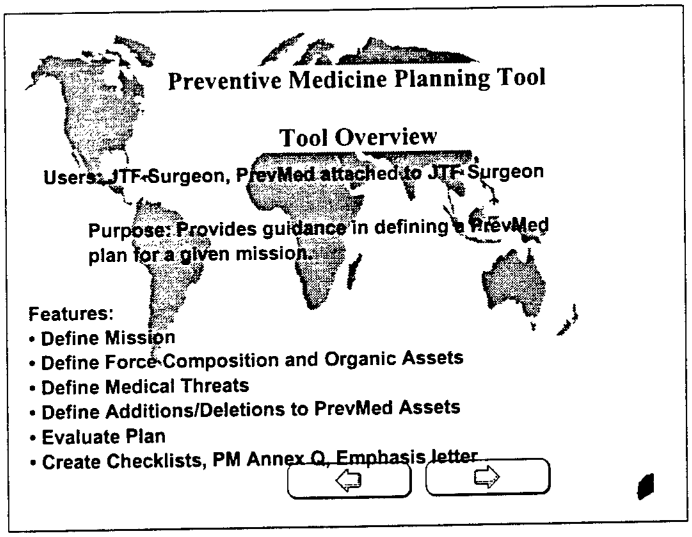


Figure 13 PMPT Visionary Demonstration Introduction Screen



Preventive Medicine Planning Tool

Tool Overview

Users: JTF-Surgeon, PrevMed attached to JTF-Surgeon

Purpose: Provides guidance in defining a PrevMed plan for a given mission.

Features:

- Define Mission
- Define Force Composition and Organic Assets
- Define Medical Threats
- Define Additions/Deletions to PrevMed Assets
- Evaluate Plan
- Create Checklists, PM Annex Q, Emphasis letter

Figure 14 PMPT Visionary Demonstration Overview Screen

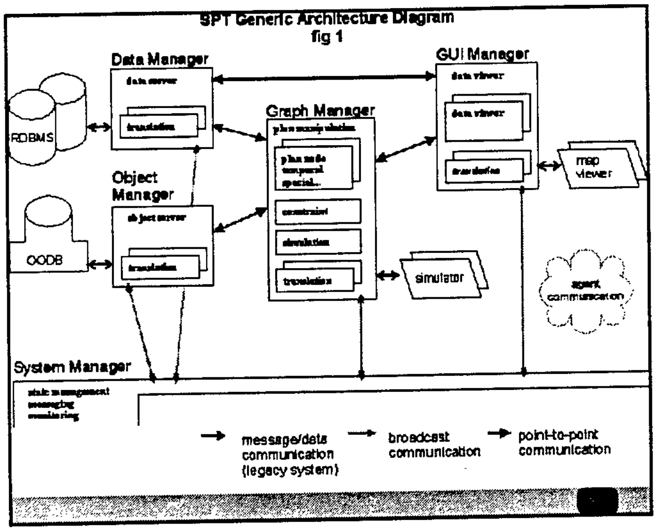
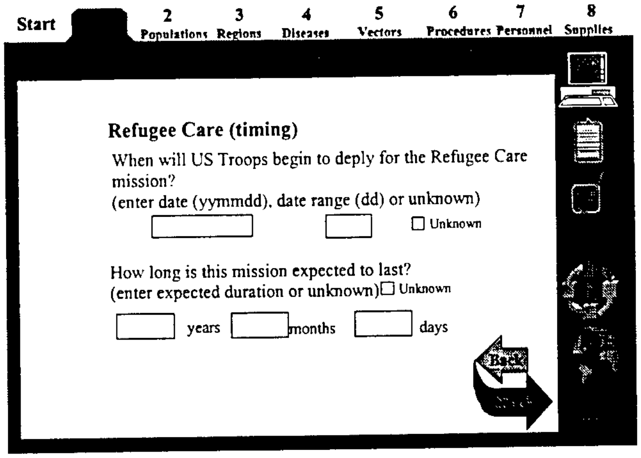


Figure 15 PMPT Visionary Demonstration SPT Architecture Screen



Refugee Care (timing)

When will US Troops begin to deply for the Refugee Care mission?
(enter date (yymmdd), date range (dd) or unknown)

□ Unknown

How long is this mission expected to last?
(enter expected duration or unknown)□ Unknown

years months days

Figure 16 PMPT Visionary Demonstration Mission Query Screen

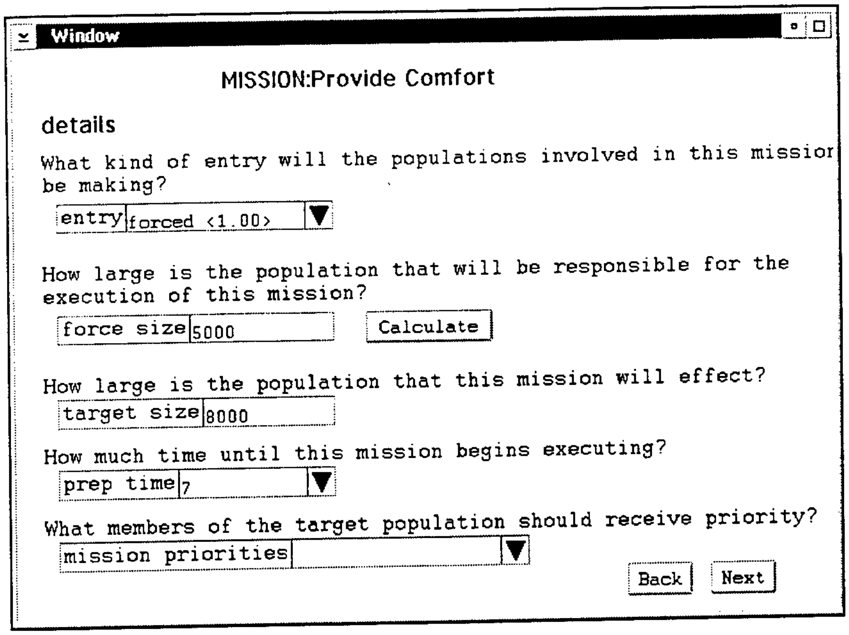
3.3.6 First Application

Using the viz-dem as a guide, the ISX team attended courses on Template's Snap product and produced an initial design for the PMPT. Due to the newness of the CTB and a PC platform requirement by NEHC, it was decided to use an OS2 platform. The combination of OS2 and Snap hosted onto OS2 limited the GUI possibilities (an example of a screen from the OS2 prototype is shown in **Figure 17 Sample Screen from the PMPT OS/2 Version**). This proved to be a major problem from the end-users' standpoints (see section 3.4.1). Part of this problem was alleviated by the Template personnel producing custom widgets for use by ISX. Once a prototype was created, it was decided to abandon the OS2 platform and the heavy reliance upon Snap and go with a Windows 95/NT platform and much less reliance upon Snap. Two major reasons for the decision were:

- End-user feedback: The end-users wanted a user interface that was much more mainstream then what was provided by the CTB. They wanted a look and feel similar to most PC applications. Further, not many Navy PCs had OS2 loaded. If the PMPT required a OS2 system, then additional PCs would be needed by the Navy along with personnel who could support those OS2 PCs. This forced aligning the application with a Microsoft development environment.

- Future support of the OS2/Snap combination: Developing in OS2 required a significant investment in programmer training and in infrastructure. At this time, IBM made a corporate decision not to target the OS2 product for the personal computer market, but instead as a server in a business environment. Thus, a business decision forced that OS2 be dropped and a Microsoft development environment be chosen.

At the time that the decision was made, the CTB wasn't supported very well in the Microsoft development environment. Thus, a small portion of the CTB was selected to continue to be used in the PMPT (i.e. the rule based decision engine) and a mainstream Microsoft development environment was used instead of the CTB.



MISSION:Provide Comfort

details

What kind of entry will the populations involved in this missior be making?

entry | forced <1.00>

How large is the population that will be responsible for the execution of this mission?

force size | 5000

How large is the population that this mission will effect?

target size | 8000

How much time until this mission begins executing?

prep time | 7

What members of the target population should receive priority?

mission priorities |

Figure 17 Sample Screen from the PMPT OS/2 Version

3.3.7 Second Application

Once the decision to proceed with a Microsoft development environment was made, the ISX development team examined how much of the old OS2 design could be used. Due to problems of switching from OS2 and Snap to a MS environment, very little could be used from the old design. Thus, the team redesigned the entire PMPT based upon the lessons learned from the first application. The redesigned system was much more acceptable to the end-users, but cost another 6 months in schedule. Examples of the screens from the new application are shown in **Figure 18 Start-up Screen from the PMPT Windows Version**, **Figure 19 Location Editing Screen from the PMPT Windows Version** and **Figure 20 Asset Editing and Force Lookup Screens from the PMPT Windows Version**.

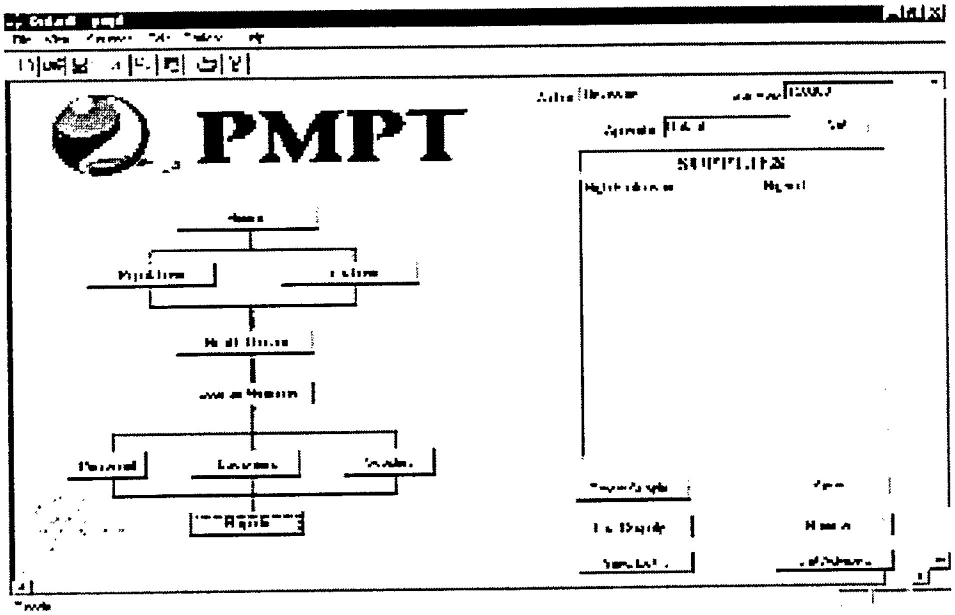
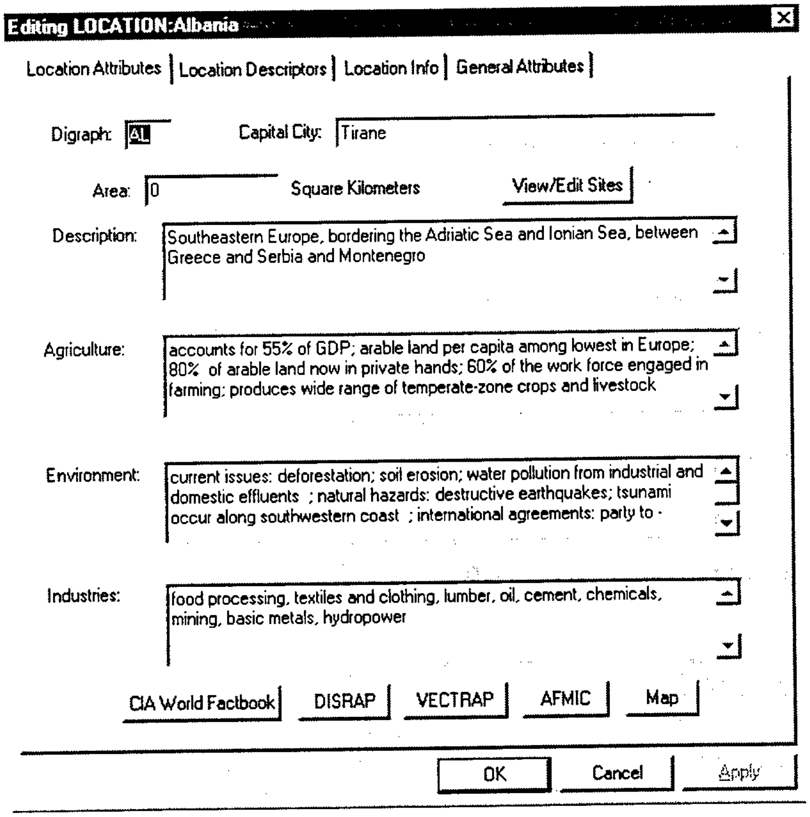


Figure 18 Start-up Screen from the PMPT Windows Version



Digraph: AL

Capital City: Tirane

Area: 0 Square Kilometers

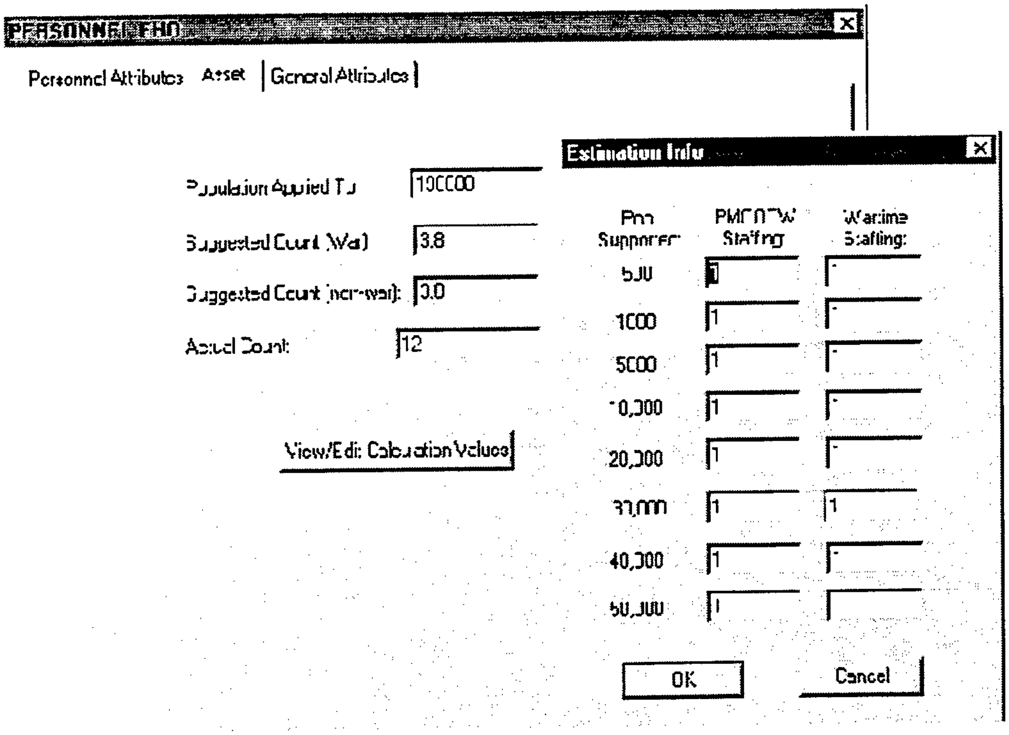
Description: Southeastern Europe, bordering the Adriatic Sea and Ionian Sea, between Greece and Serbia and Montenegro

Agriculture: accounts for 55% of GDP; arable land per capita among lowest in Europe; 80% of arable land now in private hands; 60% of the work force engaged in farming; produces wide range of temperate-zone crops and livestock

Environment: current issues: deforestation; soil erosion; water pollution from industrial and domestic effluents ; natural hazards: destructive earthquakes; tsunami occur along southwestern coast ; international agreements: party to -

Industries: food processing, textiles and clothing, lumber, oil, cement, chemicals, mining, basic metals, hydropower

Figure 19 Location Editing Screen from the PMPT Windows Version



Population Applied To: 100000

Suggested Count (War): 3.8

Suggested Count (non-war): 3.0

Actual Count: 12

Pop Supported	PMCOTW Staffing	Wartime Staffing:
500	1	
1000	1	
5000	1	
10,000	1	
20,000	1	
30,000	1	1
40,000	1	
50,000	1	

Figure 20 Asset Editing and Force Lookup Screens from the PMPT Windows Version

3.4 Lessons Learned

This section goes over the lessons learned as part of the ISX development effort. The bulk of this section is extracted from a report to Template in June 1996 which covered the problems encountered with implementing the PMPT first version (section 3.3.6). Note that most of the problems encountered with Snap were slated to be addressed in later Snap versions.

3.4.1 June 1996 Lessons Learned Document

3.4.2 Advantages of SOM/DSOM

- Contains many features that extend the CORBA standard, such as transaction processing features, persistent storage functionality and security.
- Provides inter-orb operability (IIOP)

3.4.3 Problems with SOM/DSOM

- Has a steep system dependent learning curve for installation and setup.
- Currently only runs on OS/2 and AIX and neither of these operating systems provide functionality required by our customer.
- Restrictions imposed by OS/2 include:

1. Lack of API libraries for accessing structured text data sources (e.g. Adobe Acrobat, MediaViewer, DynaText, ODBC)
2. Severely limited choices for support and development software
3. None of the ISX SPT software has been ported to OS/2 (e.g. Query Mediator)
4. Can not dynamically communicate with CD data sources via OLE
5. Difficult to install and configure the operating system (days vs. hours for NT)
6. Non-preemptive multi-tasking causes hanging and crashes

3.4.4 Advantages of SNAP

- Has allowed for the rapid creation of a prototype system.
- SNAP includes ways to indirectly reference and manipulate classes, objects and attributes that has proved useful.
- A "tab card" widget for modifying objects was provided by SNAP and has proved very useful. This widget is still experimental and needs some refinement.
- Code developed using SNAP is cross-platform and can be easily ported to other operating systems. We will experiment with this feature by porting the PMPT to SNAP running under Windows NT.

3.4.5 Problems with SNAP

- Custom relational attribute modification dialogs are better supported in 7.0, but are still cumbersome.
- No built in hyper-text help supported (plans for HTML support are being considered)
- No support for "wave over" or help text or "tool tips" (planned for SNAP 7.5)
- Spin control and pull down lists do not function or are not available (not planned)
- Some of the development features found in commercial PC software development systems that are inadequate in the SNAP development environment include:
  1. The debugger is cumbersome and inadequate for easily visualizing program execution (scheduled to be improved in SNAP 8.0)
  2. A separate debugger must be used to visualize C code
  3. A secondary editor is required to write code, the integrated code editor is inadequate (scheduled to be improved in SNAP 8.0)

4. Graphic objects can not be copied and pasted between two canvases (Change request submitted)
5. No online help/examples are available, only online reference material.

3.5 Strategic Planning Toolkit

3.5.1 Domain Description

In late-1994, a goal was established for ISX to develop a generic toolkit which would enable rapid development of strategic plan authoring tools for new domains. The idea was to draw upon the knowledge and experience gained through efforts in the areas of advanced planning and objective based planning systems, and develop a component set which would form the basis of a toolkit for strategic plan authoring.

In March of 1995, Gary Edwards and Glen Reece, Ph.D. began to analyze a sampling of domains which required or could benefit from strategic planning. This was to determine how well the existing Air Campaign Planning Tool could be transitioned to those domains in a generic way as to be widely applicable. Nine potential domains were evaluated to a greater or lesser degree and from this evaluation a set of base requirements were established. These are:

1. The use of simulation to assist with understanding plan sensitivity issues.
2. Provide the support necessary to project different potential outcomes and understand their likelihood.
3. Provide the support necessary to make resource constraint projections and understand the incremental benefits of plans that more aggressively utilize resources versus ones with less aggressive utilization.
4. Provide the ability to identify side-effects of an action(s) and to be able to constrain those side-effects.
5. Provide the ability to quantify objectives so that the user can observe expectations from simulation versus actual data as the plan unfolds (we need a well defined method to develop metrics associated with objectives).

The main concern for any domain to which the SPT is to be applied is whether we can gather the necessary information into SPT so the appropriate decisions can be made. This will have to be determined in USE sessions as we become more serious about applying SPT to a particular domain. Due to other obligations, this effort was put on the back burner after this initial session.

In August 1995, Richard Myers brought a group together in Westlake to discuss SPT architectural issues. This meeting was essentially a session in which the ISX personnel that were intimately familiar with the ACPT were interrogated about how they perform the task of building strategic planning tools and were asked to describe what facilities the SPT would have to provide to improve this process.

In 1996 we again focused on the SPT concept. These efforts, as well as, our work in developing a strategic planning tool for medical readiness have lead to the development of the SPT Core (the generic class model for the SPT).

3.5.2 System Vision:

The goal of the SPT Project is to produce a suite of software components which will enable the rapid construction of strategic planning systems for a wide variety of domains addressing an equally wide variety of functional requirements. This means that many small software components will be designed and implemented with well-defined interfaces which can be plugged together with other components to provide required capabilities.

We want to be able to build upon our previous development efforts and apply our knowledge and expertise in advancing the state-of-the art of strategic planning tools and not have to spend so much time on reimplementation because we (1) didn't "expect" to reuse developed capabilities, (2) we didn't have time to develop a generic capability, or (3) we didn't plan to have to run the system on another platform.

The SPT will enable us to chose from a "component library" those capabilities we need for implementing a strategic planning system for a particular domain and focus on developing new capabilities to exceed the customer's expectations while giving us a new set of capabilities to add to the component library for future efforts.

3.5.3 Status

The TRP SPT effort was focused in four main areas as a first step towards developing a component library. The focus areas for the project were (1) Data Access Services, (2) Authentication Services, (3) SPT Core, and (4) Management Services. The results of each of these areas are described in greater detail below.

3.5.4 Data Access Services

A vital component of SPT Suite or component library is that of a service for allowing and managing data access. The type of services we envision is modeled upon the I3 Service Architecture which were Query Services, Source Services, and Integration and Transformation Services. These services come together to provide a mediation capability for repositories of information. In this effort we focused on how to enhance the core capabilities provided by ISX.

The first area was in evaluating existing systems to determine whether our existing capabilities could be augmented by adopting a specific approach or extrapolating from a particular approach. The two systems which were examined were SIMS (from USC Information Sciences Institute) and HERMES (from the University of Maryland).

The second area was in review of our Source Service code and libraries to determine how best to restructure them to support the development of plug-and-play components such as those we envisioned for the component library. This effort lead to a redesign of the code structure and library hierarchies.

3.5.5 Authentication Services

In several other previous projects there was a ubiquitous requirement to provide controlled access by users to the system and to specific capabilities of the system. We saw this as a prime example of the type of capability which could be provided in a component library and used across a wide variety of systems regardless of the specific domain. Therefore we designed and implemented a CORBA-based component for user authentication called the Account Manager.

3.5.6 SPT Core

Design and initial implementation of the SPT Core model was based upon the STT paradigm. The majority of the STT functionality is embedded with the SPT Core. This functionality included the ability to represent the various strategy levels, the relationships (e.g., dependencies) between the strategy levels and the nodes (the tasks), abilities to evaluate the completion of a task, the roll-up of the completion statuses to an overall status, etc. This effort also developed the necessary test databases, tools for populating the databases, error logging, etc.

3.5.7 Management Services

Similar to the ubiquitous requirement of having authentication services for a wide range of systems, there is likewise a requirement to provide a capability to manage contact information for related and responsible parties. We saw this as another example of the type of capability which could be provided in a component library and used across a wide variety of systems regardless of the specific domain. Therefore we designed and implemented a CORBA-based component for contact management called the Contact Manager. We did not complete the entire implementation, as the client side was not implemented, but the server is fully functional under Windows NT 3.51 (and would require little effort to migrate it to another platform).

3.6 Medical Waiver

Towards the end of the contract's period of performance, IBM had some money left over which was distributed to the other three contractors. ISX used the funds to continue support on the PMPT but also to explore another application. One criteria on the consortium's part, was to use the CTB to a much greater extent then was done with the PMPT. After some discussion with the Navy's Bureau of Medicine and Surgery (BuMed), it was agreed to look at the Navy's Medical Waiver process. This process was fairly well defined, could use automation, had a progressive-minded point of contact (CAPT Warren Jones), and lent itself well to Snap's WorkFlow tool. Due to the very limited amount of money available for this effort, it was agreed with CAPT Jones to implement as much functionality as possible, but no promises were made and with the understanding that a completed, hardened tool wouldn't be possible under these funds.

3.6.1 Domain Description

The Medical Waiver process allows individuals who don't pass a Physical Examination (PE) for a position for which they are applying[ii], to be allowed to continue pursuing that

[ii] The Physical Examinations are administered when a person is making a major transition. Some situations when the PE is required is when a person enlists, when a person applies for Officer Candidate School,

position. A person could be granted a waiver if their job category is in high demand, their job category doesn't require full physical abilities, etc. Thus, the Waiver process allows the Navy to "stretch the rules" when the stretching doesn't affect the ability to get the job done.

The Medical Waiver process is diagrammed in **Figure 21 Navy Medical Waiver Process**. There are multiple places where BuMed would receive a waiver application and then issue their recommendation. The entire waiver process lends itself well to Template's WorkFlow tool. The initial assessment, which was later proven by the application development, showed that all the application development could be done within the WorkFlow development environment. The tool allowed the creation of user types (e.g., Applicant, Waiver Recommendation Reviewer), their corresponding interface screens, the documents and the document flow between the end-users/organizations. Unfortunately, TRP time and money did not allow for a completion of the demonstration application which is now pending additional funds from BuMed.

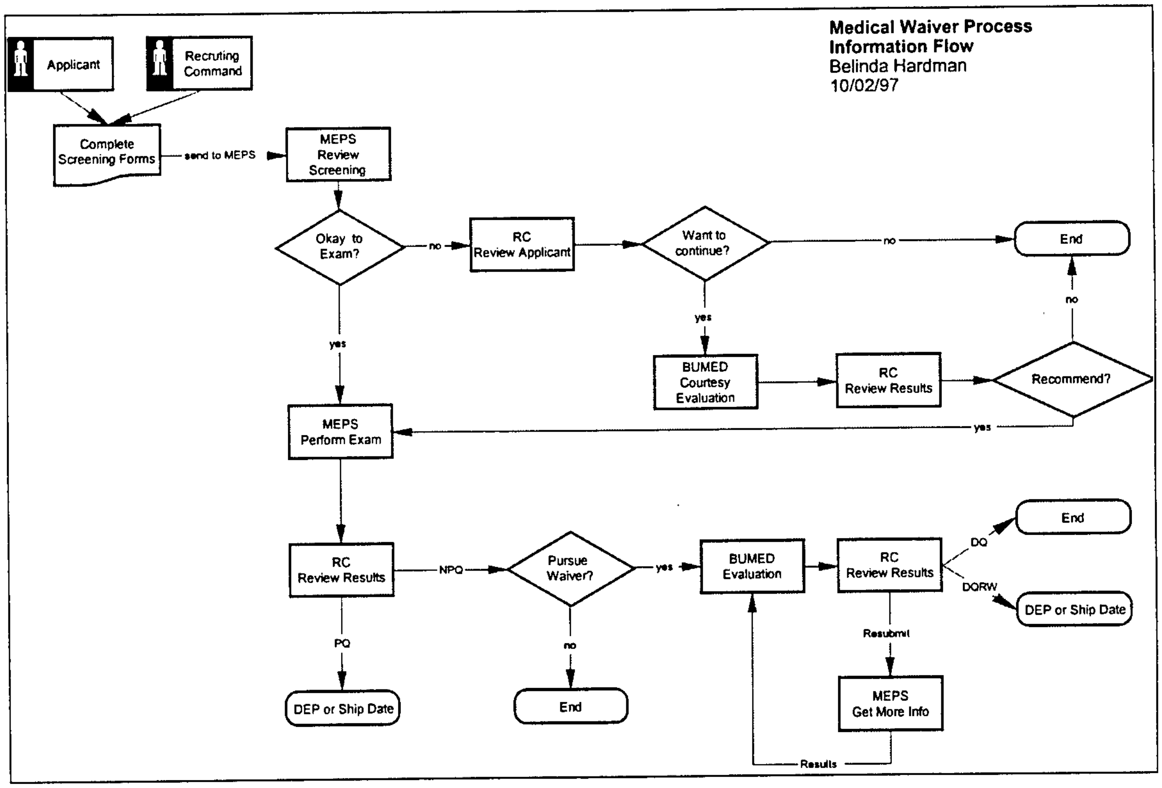

Medical Waiver Process
Information Flow
Belinda Hardman
10/02/97

Figure 21 Navy Medical Waiver Process

3.6.2 Summary

ISX's major role in the TRP was to provide a proof of concept application of the CTB to military applications and to feed in military-domain requirements to the CTB. Two applications were developed to fulfill this. The first application was the Preventive

when a person graduates from the Naval Academy, etc. Situations where a PE is not administered is when a person has a normal grade promotion, during the course of their normal work assignment, etc.

Medicine Planning Tool upon which the bulk of ISX's effort was spent. The development of this tool initially used as much of the CTB as possible. However, due to user requirements and the unavailability of certain features when needed, only a small portion of the CTB was used in the final PMPT. Feedback was generated by ISX during the application development stage and given to Template and IBM. These companies did use this information in making decisions on their product's features with many of the PMPT developer's "wish list" becoming part of the CTB.

The second proof of concept application was developed towards the end of the TRP. This application applied one of Template's products to automating the Navy's Medical Waiver process. With limited time and funds, only a visionary demonstration was attempted. This effort did show that all of the end user requirements could be met by using the CTB.

Overall, the TRP accomplished what it set out to do; namely, creating a software development environment which shortens the development time and is applicable to military domain projects.

The next page is a description of the PMPT used at the NEHC Conference in Virginia Beach in February 1997.

Preventative Medicine Planning Tool

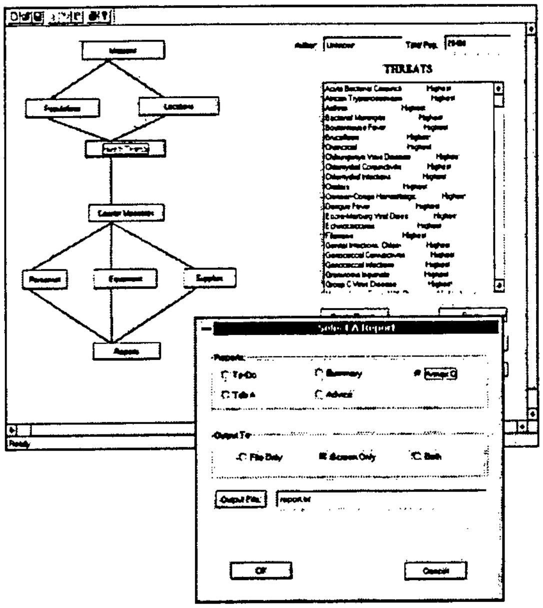
Technology that works the way a Preventative Medicine professional does:

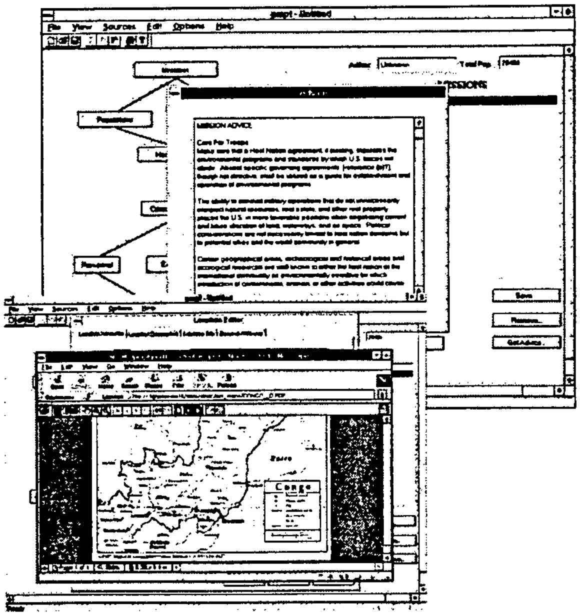
- Assists by ensuring mistake avoidance
- Provides guidance throughout the PM planning process
- Identifies impacts of plan changes
- Incorporates PM calculations for personnel, equipment and supplies
- Reduces administrative burden by generating PM Appendix 8 to Annex Q for OPLAN, to-do lists and draft versions of CO Emphasis letters, and Line Letters.
- Provides single point of access to information from many sources including MEDIC, CCDM, DISRAP, VECTRAP, Maps, and the Internet.
- Provides PM mission guidance and advice to the Medical Planner through the use of embedded rules based upon previous experience (i.e., lessons learned).

Addresses the full spectrum of PM missions for war and military operations other than war.

State of the art Technology

- Graphical User Interface that allows easy manipulation of information and interaction
- Standards-based client/server tool implemented in C++, Snap™, and SOM/DSOM (CORBA).
- Takes advantage of existing COTS products to display information in a variety of formats.
- Offers a wide range of data and information management tools including Cut & Paste features from any information source, and pop-up advice (based upon historical operations, PM





guidelines, and unique situations and lessons learned).

DASH'R TRP
Final Report
HONEYWELL
April-95 through November-97

By: Robert Goldman

4. Honeywell

In this introduction, we briefly summarize the final report. We begin by explaining Honeywell Technology Center's role in the DASH'R OTRSD Consortium, that of providing civilian validation of the consortium's dual-purpose object technology. This validation was done by developing a civilian advanced software scheduling application for batch manufacturing (itself a dual-use manufacturing technique). Accordingly, we provide a brief introduction to batch manufacturing and the corresponding scheduling problem. We then present our objectives—the features we identified as essential for this application. We met these objectives by using a proprietary scheduling technique we call constraint-envelope scheduling, which we introduce in the following section. We then discuss our development method and the course of development we followed in this program. Finally, we conclude by outlining the rest of the document, which provides further details on all these topics.

4.1 HTC Role in Consortium

Honeywell Technology Center (HTC) has provided civilian validation of the dual-use technology developed by Consortium members Template Software and IBM. HTC has demonstrated the consortium object technology by using it in the construction of a commercially available scheduling system for batch manufacturing. Batch manufacturing is an increasingly important form of manufacturing that is a hybrid between discrete and continuous manufacturing. Batch manufacturing is important for both civilian and military production. In the following section, we give a brief introduction to batch manufacturing and the corresponding scheduling problem (more details are given in the body of the document).

Batch manufacturing, a dual-use manufacturing technique, occupies the gray area between continuous and discrete manufacturing. In continuous manufacturing, a stream passes through a process, yielding a continuous stream of product. In discrete manufacturing, an identifiable individual product is created through a series of steps in a plant. In batch manufacturing, there are neither individual products nor a stream of product; rather, one makes product in batches.

Some sample batch manufacturing processes are the making of food mixes, composite materials (e.g., for airplane control surfaces), and beer brewing. These show the wide range of complexity within batch manufacturing. Food mix manufacturing is often largely a matter of materials handling: getting the right mix of substances into a container and stirring. On the other hand, both composite materials manufacture and beer brewing involve complex chemical reactions.

Batch manufacturing was chosen as an application for this TRP for several reasons. First, batch manufacturing is an important dual-use technology. Advanced batch manufacturing techniques are widely used in the aviation industry for building aircraft components out of advanced composite materials. Second, a batch manufacturing scheduler provides an important opportunity for commercialization. Although there are a wide variety of programs available for scheduling discrete manufacturing processes and planning for continuous processes, few, if any, scheduling products are capable of tackling the distinguishing features of batch manufacturing (e.g., tank handling, variable changeover times). Finally, HTC had already developed a prototype Batch Manufacturing Scheduler suitable for reimplementation according to the purposes of this TRP. The previously existing prototype was ill-suited for productization because it was implemented in the Common Lisp programming language, which is rarely used in the software industry. However, this prototype did tackle the full range of batch manufacturing phenomena and was written in an object-oriented style.

The scheduling problem for batch manufacturing is different from the corresponding problem for discrete manufacturing, hence the need for a special-purpose scheduling system. Scheduling systems for discrete manufacturing are typically based on the *job-shop scheduling* problem (i.e., jobs must be scheduled for exclusive use of workstations in the plant). Although scheduling workstations is an important part of batch manufacturing scheduling, it is only a part. A full solution to the batch scheduling problem must also take into account the connections between workstations in a plant, since typically batches of product will have to be moved from workstation to workstation through a restricted network (e.g., a set of pipes or headers). Properly handling this problem requires us to model the hierarchical structure of batch recipes (the procedures by which products are manufactured) and the topology of the plant floor. Another important issue in batch manufacturing is storage management. In many batch plants, products must be stored in sets of tanks. The proper use of these tanks—avoiding overflow, underflow, or time-consuming cleaning processes—is a critical issue for batch manufacturing that does not arise in discrete manufacturing. Finally, we must handle the relationships between the various jobs in the plant. In batch plants, complex relationships often exist between consecutive jobs on a particular piece of equipment. For example, in a food plant, one always wishes to make light-colored foods before dark-colored foods, because this can be done without having to clean the equipment; if a product is made using eggs, the equipment must be steam-cleaned before it can be used again, so it is better to make one large batch of mayonnaise rather than two smaller batches.

4.2 Objectives

Our objective in this program was to develop an integrated batch manufacturing scheduling (BMS) system and prepare it for commercialization. A secondary objective was to develop a version of HTC's library of scheduling algorithms and object classes that would provide a foundation for fielding scheduling systems for a wide variety of markets.

For HTC to meet its objectives in BMS development, (1) the resulting system must be suitable for incorporation into the existing organization of batch manufacturing enterprises and (2) the BMS must be built using software techniques that are readily maintained by Honeywell software engineers.

We have identified two requirements for a scheduling system to be of value to a customer's enterprise. First, it must fit within the enterprise's organizational structure. In the case of batch manufacturing enterprises, the scheduling system must be capable of working with enterprise planners and with plant operations personnel. Enterprise planners typically use some form of materials requirement planning (MRP) system to manage inventory and meet customer orders. Plant operations personnel are responsible for meeting the production orders issued by enterprise planners. The key tool used by plant operations personnel is the distributed control system (DCS). We discuss these interface requirements in greater detail below. The scheduling system provides a point of contact between business concerns and operations. To provide that point of contact, the scheduling system must be *distributed.* It must be possible for multiple parties in a given enterprise to interact with the BMS; it is not sufficient that it be a desktop tool supporting only a single access.

The second requirement for a useful BMS system is that it must provide *and maintain* schedules that meet key constraints. In the case of batch manufacturing, the key constraints are exerted by the structure of the plant, supply of raw materials, production orders, and product recipes. We stress the maintenance of schedules as artifacts because this has been a weak point of previous scheduling systems. Previous scheduling systems have concentrated on providing an initial schedule and have ignored the problem of schedule maintenance.

It is not sufficient to design and develop a useful software application; we must also get it into the hands of users. The Honeywell Technology Center is not a product development center. Rather, HTC provides R&D services to the Honeywell divisions (Home and Building Control, Industrial Automation and Control, and Space and Aviation) and, under contract, to external customers. HTC does not have the facilities to support and market software applications. Accordingly, it is critical that software systems developed at HTC be built in a way that fosters technology transfer.

The DASH'R OTRSD Consortium technology provided HTC an opportunity to take an existing BMS prototype system and reimplement it in a form more suitable for transfer to the Industrial Automation and Control (IAC) division of Honeywell. In particular, we used the object-oriented enterprise integration framework supplied by Template Software, together with CORBA, to provide a distributed application framework suitable for integration into enterprise information systems. At the same time, we took the core facilities of the existing BMS system and translated them into conventional object-oriented languages (C++ and Java) for maximum efficiency, portability, and maintainability.

4.3 Scheduling Technique: Constraint-Envelope Scheduling

In our approach, which we call *constraint-envelope scheduling (CES)*, schedules are constructed by a process of "iterative refinement" in which scheduling decisions correspond to constraining an activity with respect to either another activity or to some timeline. The schedule becomes more detailed as activities and constraints are added.

Undoing a scheduling decision means removing a constraint, not removing an activity from a specified place on the timeline. CES is a *least-commitment* approach. We do not assign a set of activities to places on a timeline, assigning each activity a start and end point. Rather, we collect sets of activities and constrain them only as needed.

The least-commitment nature of our schedules is an important advantage when it comes to rescheduling. If an event arises that makes a resource unavailable, or an ongoing activity takes longer than expected, the effect on the schedule is minimized. First, only those activities related by a chain of constraints to the activities explicitly moved will be affected. Second, if the set of constraints in the schedule are consistent with the new event, the projected effect of the schedule can be updated efficiently, without any rescheduling at all. This is how we implement the distinction between schedule modification and rescheduling, discussed in the previous section. For example, if one step of a complex process takes longer than expected, but the process (and the other processes that follow it) can still be completed before their deadlines, then all that needs happen is to update the estimated start and completion times—a task that can be accomplished very quickly. No reordering or reassignment of activities need take place.

Our successful application of constraint-envelope scheduling relies heavily on three supporting facilities. The first is a set of sophisticated search techniques, loosely based on an extension to Ginsberg's dynamic backtracking.[iii] The second is a highly optimized temporal constraint engine. Determining the impact of a scheduling decision on a set of temporal constraints is the activity that is the "inner loop" of our scheduling approach. Our search engines must be able to explore the consequences of many different scheduling decisions. This is made possible by the effective exploitation of an extremely efficient temporal constraint manager called the *Interval Constraint Engine* (ICE). The final supporting facility is a library of object classes that provide *flexible constraint objects* (FCOs). The FCOs stand between a domain model, which represents the problem to be solved, and the search engine, which provides the means of finding a solution. Because of this intermediate layer, we need not develop new search engines for each scheduling problem; we need only supply a set of flexible constraint objects that implement a set of transactions between the search engine and the problem to be solved.

4.4 Development Method

For this project, HTC adopted an approach of *iterative development* from an existing research prototype. Over the course of the project, we attempted to develop a new version of the complete BMS system every six months. Each new version differed from its predecessor either in having a more complete implementation of existing features or in having a more complete set of features. This development technique has two major advantages. First, when developing an experimental product like the BMS, this maintains assurance of feasibility. When building an experimental product and developing in the conventional design-specify-implement framework, there is always the danger of designing a product with an infeasible set of features. Second, having a prototype to demonstrate at any time makes it easier for us to validate the system against user needs.

[iii] Matthew L. Ginsberg, "Dynamic Backtracking," *Journal of Artificial Intelligence Research* **1**, 25–46, 1993.

4.5 Course of Development

The existing BMS prototype was developed at HTC in 1994, supported by internal R&D funding. The environment in which the prototype was developed placed a premium on rapid prototyping. Rapid prototyping permitted us to develop a version of the system, display it to customers, and alter system features quickly to converge on a key set of features and algorithms to support those features. However, the tools used for rapid prototyping, object-oriented Common Lisp and the Garnet user interface prototyping system, made technology transfer difficult.

Our actual development process was not quite a pure version of iterative development. Rather than plunging directly into the development process, we went through an early design phase during which we captured the object-oriented design of the existing BMS and revised it for the new, distributed OO framework. After this, we began by translating the core facilities of CES—the search algorithm, the interval constraint engine, and the core activity and task models. Simultaneously, we were experimenting with CORBA for distributed object orientation, using the existing (Common-Lisp) batch scheduling system.

In the last year and a half of the project, the original prototype was increasingly discarded and work focused on the final system. During the final year, a significant change of direction took place. Work was proceeding too slowly on the core facilities, developed in C++. Accordingly, we decided to implement those core facilities in the new programming language, Java, rather than C++. Because our Java code development was so much more efficient, and because existing work in C++ could be translated to Java with little effort, the change in programming language did not cause a net delay to the project.

In the final six months of the project, our attention shifted to using SNAP's Workflow Template (WFT™) technology to provide an enterprise integration framework within which to deploy the BMS. We used WFT to provide communication between the three enterprise layers: the BMS, production planning (represented in the demonstration system by order management), and a plant control system simulation.

Finally, we were provided with the opportunity to extend the period of the contract and perform additional work. During this period of performance, we focused on preparing the BMS for commercialization. To do so, we concentrated on extending the applicability of the BMS from batch manufacturing to the continuous process industries, particularly oil refining.

4.6 Road Map to Document

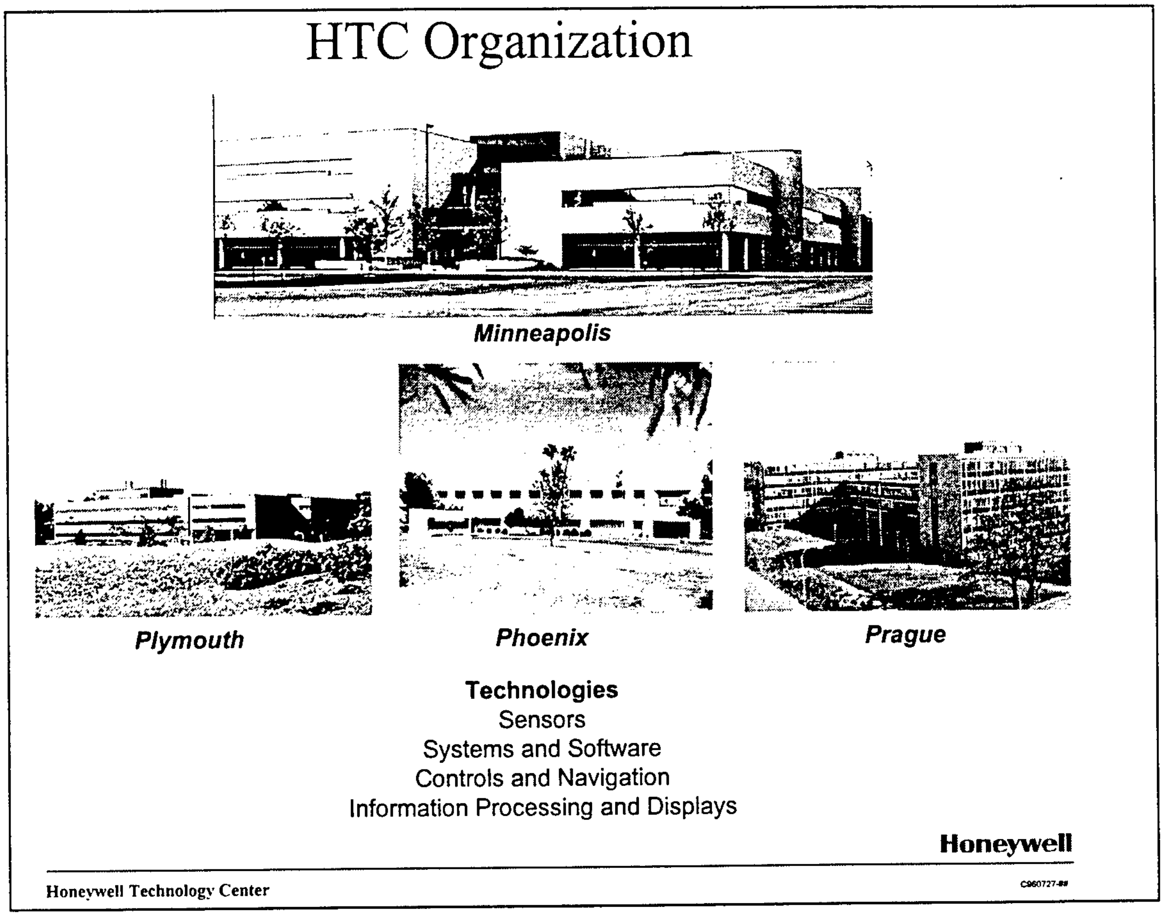
In the following section, we provide some background necessary to understand our work in this project: what sort of organization HTC is, what batch manufacturing is, and how batch manufacturing gives rise to scheduling problems. We also introduce HTC's distinctive scheduling technology, *CES*. With the background filled in, we go on to explain the requirements for a BMS. We focus on the interactions required of the BMS: with the user, with the manufacturing plant's control system, and with production planning systems. We then describe the system architecture we have developed to meet these requirements. The system architecture is based on a set of models, both generic scheduling models and specializations of these models that capture particularly distinctive features of batch manufacturing. To interact with those models in an object-oriented and modular way, we have developed a method of *flexible, object-oriented constraint satisfaction,* whose implementation has been carried through in the course of this program. After describing these key features of the BMS, we move on to give an account of the course of our development. For the reader interested in more engineering specifics, this is followed by a section that discusses the tools we used in our development process. After that, we outline our plans for making the work done on this project into Honeywell products. We conclude with a brief summary.

4.7 Background

4.7.1 Honeywell Technology Center

Honeywell has built its long-standing reputation as a leader in control systems by balancing investment in visionary research and integration of state-of-the-art technologies. This historic strength is reflected in the diverse technology areas that comprise its core business, including aircraft flight management and avionics, industrial plant control, spacecraft and satellite control, and home environment control.

HTC is an organization of over 300 scientists and engineers based in the Minneapolis/St. Paul metropolitan area (See **Figure 22 Honeywell Technology Center**). HTC also has satellite organizations in Prague and Phoenix, Arizona. HTC is the company's primary research and development organization, delivering the advanced technology, processes, and product and service concepts needed to satisfy Honeywell's aerospace, industrial, and home and building control customers worldwide. HTC maintains a broad technology base in controls, systems and software, information processing and displays, and sensors. HTC technology developments are currently responsible for hundreds of millions of dollars in Honeywell revenues each year. About half of the research performed at HTC is funded by federal agencies, including DARPA, NASA, NIST, the FAA, FHA, and the DOE. Additional support comes from Honeywell divisions. The HTC web site http://www.htc.honeywell.com is currently under construction but does provide a high-level overview of the research center.



HTC Organization

Minneapolis

Plymouth

Phoenix

Prague

Technologies
Sensors
Systems and Software
Controls and Navigation
Information Processing and Displays

Honeywell

Honeywell Technology Center

C960727-##

Figure 22 Honeywell Technology Center

Technology transfer is a charter task of HTC. Concept development and demonstration is just the first step in any truly successful development program. The transfer and implementation of any technology to industry in support of military and commercial business opportunities is the true test of the value of the technology and the commitment of the participants.

HTC's corporate charter is to transfer leading-edge technology to Honeywell customers, including Honeywell product divisions, related industrial partners, and military prime contractors. HTC's technology transfer success stories include the highly successful Boeing 777 integrated avionics, dual-use ring-laser-gyro product family, Ada, and VHSIC, to name a few. Because HTC supports both military and commercial product divisions, we have demonstrated dual-use technology transfer for more than 10 years. In addition, HTC has a proven record of technology transfer to both industrial and academic communities. We continue this tradition of open community technology transfer in the current DARPA-funded programs DSSA, RASSP, and Prototech.

The work performed under this contract was primarily done within the Automated Reasoning group at HTC. The Automated Reasoning group consists of 16 research staff, plus supporting personnel, within the Systems and Software technology area. We have ongoing projects in the areas of planning and scheduling, constraint-based design, simulation, health maintenance and diagnosis, real-time AI, information modeling, neural nets, data mining, and collaborative agent-based systems. The work environment fosters the development of individual research projects either within or complementary to current research areas. The research we conduct tends to be applied. Proving out ideas through the construction of prototype systems is a common activity, though not to the complete exclusion of more theoretical research. Interaction with the larger research community and collaboration with researchers both inside and outside of Honeywell are encouraged

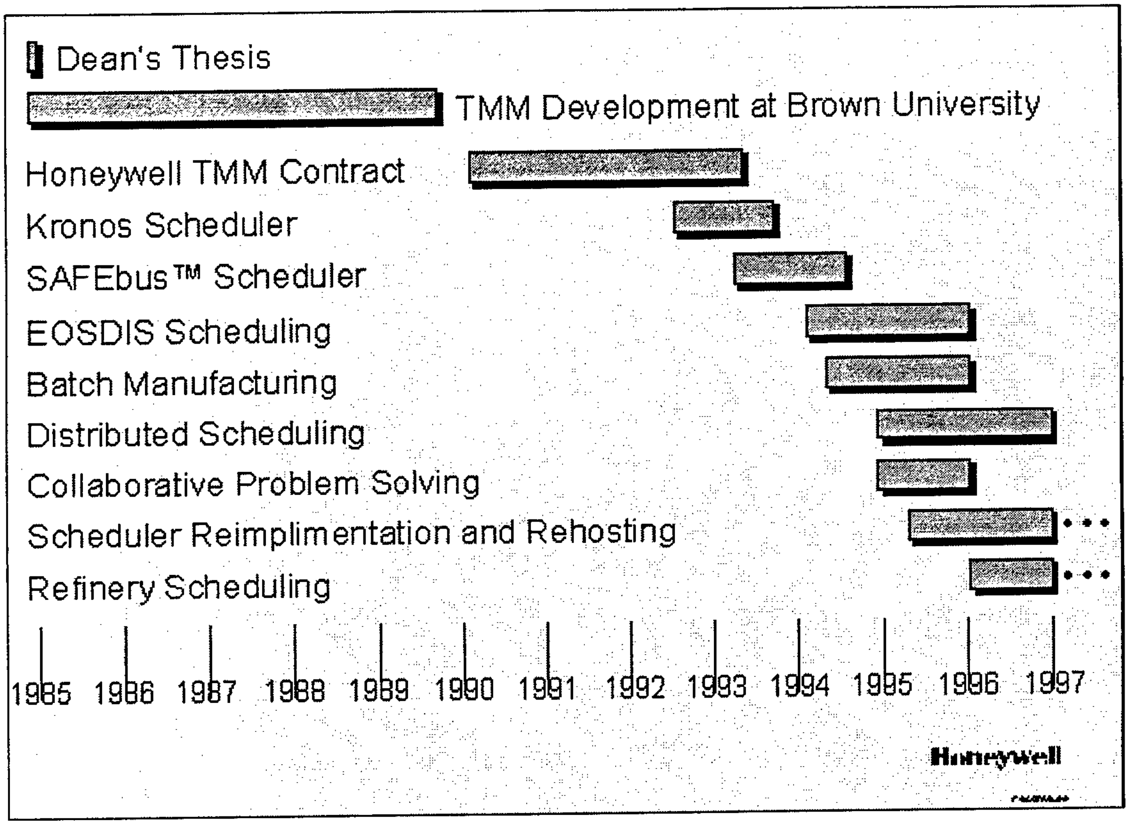


Figure 23 History of Scheduling at HTC

Scheduling has long been a key thrust area of the Automated Reasoning group. This thrust area has grown out of an initial DARPA contract to develop a version of the Time-Map Manager (TMM). Scheduling tools based on the TMM technology have been developed for a wide variety of domains, including operations planning for a Space Shuttle science module, satellite data analysis and retrieval for NASA's Mission to Planet Earth, processor and communication scheduling for the Boeing 777 Flight Management System, and batch scheduling (see **Figure 23 History of Scheduling at HTC**). These systems run the gamut from predominantly manual to completely automatic.

Spacehab. The *Kronos* scheduler was developed for the purpose of assisting humans in scheduling experiments for a science module that has flown on several Space Shuttle missions. The scheduling process is largely manual. Activities are manually selected to be added to the schedule. Constraints such as execution windows and predecessor/successor relationships are enforced automatically. The availability of needed resources is also checked automatically—a real benefit, since resource requirements can be complex—but the choice of resource assignment is left to the human user, unless there is only a single consistent assignment. Possible and necessary resource conflicts are detected and presented to the user for resolution, either through the addition of ordering links between activities or by changing resource assignments. These "resource bounds" are the visible manifestation of the envelope of schedules consistent with the current set of constraints.

EOSDIS. NASA's Earth Observing System (EOS) is a multiyear, multibillion-dollar project aimed at gathering scientific information about the Earth's environment through satellite-based remote sensing. The EOS Data and Information System (EOSDIS) will be responsible for archiving and analyzing the resulting data. EOSDIS functions include managing mission information, archiving and distributing data, and generating and disseminating scientific data products. We have provided two separate applications in support of this process. The first is the *Data Archiving and Distribution Scheduler* (DADS), which schedules the retrieval of data from terabyte tape archives, its staging on disk during processing, and eventual distribution to the interested parties. DADS includes an interface to an execution and dispatch system through which the schedule is updated automatically as the course of events diverges from that predicted in the current schedule. This updating capability highlights one of the strengths of the least-commitment CES approach. As long as the course of events does not diverge too far from that predicted, the system simply updates the projected start and end times of activities as appropriate. When a deadline or some other commitment is threatened, that fact is detected and the system will reschedule, either automatically or by prompting a human user to make a decision.

The other system we have provided is an extension of an existing NASA planning tool for image analysis, based on NONLIN. Our extension of this tool in a system called PLASTIC adds the ability to backtrack in planning based on estimates on the duration of planning (sub)tasks at any level in the planning hierarchy. These duration estimates are ultimately derived from worst-case bounds on execution times for the primitive subtasks involved, but can also be modified directly based on previous experience. The estimates are updated automatically as the system runs, permitting a smooth response to a changing computational environment.

Avionics scheduling. Another of the applications to which we have applied CES is the generation of static schedules for processing time and bus communications, involving safety-critical applications running on flight hardware on a commercial airplane. The schedule is static for reasons having to do with verifiability and repeatability of behavior, and ultimately with FAA certification for flight safety. This problem is both large and complex. In a typical problem instance, there are approximately 30,000 activities. There are six processors, all between 80% and 90% loaded, with processes periodic at rates between 5 and 80 Hz. Data communication is specified between processes, not between process instances. To make matters worse, we are constructing a schedule for a 200-ms "frame" that itself runs at 5 Hz. Communication from one instance of this frame to the next is entirely legal, so there is in some sense a circular model of time in which constraints on activities late in the frame may affect activities early in the frame.

The system constructed for this problem generates an initial schedule entirely automatically—the problem is too big for any but the most abstract level of manual intervention. Rescheduling is a more interactive process in which the user may specify preferences regarding what in the existing schedule will change. These preferences are structured according to how changes in the schedule affect the certification level of various system functions.

4.7.2 Batch Manufacturing

Batch manufacturing occupies the gray area between continuous and discrete manufacturing. In continuous manufacturing, a stream passes through a process, yielding a continuous stream of product. In discrete manufacturing, an identifiable individual product is created through a series of steps in a plant. In batch manufacturing, there are neither individual products nor a stream of product; rather, one makes product in batches.

Some sample batch manufacturing processes are the making of food mixes, composite materials (e.g., for airplane control surfaces), and beer brewing. These show the wide range of complexity within batch manufacturing. Food mix manufacturing is often largely a matter of materials handling: getting the right mix of substances into a container and stirring. On the other hand, both composite materials manufacture and beer brewing involve complex chemical reactions.

Batch manufacturing processes share several common features. Batch processes are characterized by *relatively* frequent changes from product to product. We typically see more frequent change of product than in continuous manufacturing, where one attempts to maintain stable conditions, but less than in flexible discrete manufacturing, where one sees production of customized single products (e.g., cars). Control of batch processes requires a combination of both continuous and discrete process control. One feature often found in batch processes is limitations on the amount of production in a single batch, imposed by vessel sizes. For example, in beer brewing, the amount of brew that can be made in a batch depends on the sizes of the kettles in the brewery.

Batch processes are performed according to procedure descriptions called *recipes*. Recently, recipe terminology has been standardized by the Instrument Society of America (ISA). Recipes are hierarchically organized. There are three levels of recipe. The highest level is the *general recipe* that describes the make up of the product. For example, in a reactive process,[iv] this might be a description of the set of operations necessary and the proportions of the various ingredients. At the next lower level, the *site recipe* describes how a product is made at the particular plant in question. One of the most significant pieces of information at this intermediate level is a description of the *process train*: the set of plant *units* that the batch will pass through. At the lowest level is the *unit recipe*—the set of operations that will be executed by a particular piece of the plant. For example, a chemical process might have two unit recipes, one for a premixer that mixes the set of ingredients and ensures that the proportions are correct and one for a reactor that is charged by the premixer (and possibly with some catalysts as well), heats and agitates the mixture, and finally dumps it to a storage tank.

[iv] We use the term "reactive" to mean a process that involves chemical reaction, such as composite curing or beer fermentation. This is distinguished from nonreactive processes such as mixing, used in making, for example, powdered food mixes.

4.7.3 Batch Manufacturing Scheduling Problem

4.7.3.1 Elements of the scheduling problem

Batch scheduling is the problem of fulfilling a set of *product orders*, which must be made according to some *recipes*, within the limitations of a particular *plant* and subject to a set of *constraints.*

Orders. The orders specify substances to be produced, amounts and deadlines, and release times. These represent the request that a certain amount of the substance in question be produced by the deadline. The release time indicates the earliest time production can start; this is often necessary for products that can spoil. The production orders need not correspond one-to-one to customer orders. In a just-in-time (JIT) manufacturing plant, production orders are only issued in response to customer orders, so for each customer order, there will be a production order for each product the customer wants. However, most plants today are not pure just in time plants. Instead, most plants try to keep a particular size inventory of each product. Often the size of this inventory is determined based on projected demand, according to an MRP regime (see discussion of interfaces, below, 4.14).

Recipes. Each product has an associated set of recipes (for most plants there will be only one recipe per product). Each recipe defines how the activity of producing a batch of product is decomposed into a set of individual activities. The recipe will also have associated with it a nominal production amount and a minimum and a maximum production amount. The minimum and maximum production amounts will typically be determined by the capabilities of key pieces of equipment. For example, in a brewery, the size of a batch of beer will typically be limited by the size of the kettles in the plant. If the largest kettle is 1000 gallons, that will be the size of the largest batch.

Each of the activities in the recipe will have an associated set of *resource requirements* and a *duration.* Typically, the resource requirements will include a piece of plant equipment that the activity will need to have for its exclusive use. For example, when fermenting beer, one must have a tank exclusively devoted to fermenting the batch in question. There may also be ancillary requirements. Typical ancillary requirements include the use of *metric resources* shared across the plant. Depending on the plant, labor may be an important ancillary requirement. In reactive processes, plant utilities such as steam heat and electrical power may be limiting factors. The duration of the activity may be a function of the size of the batch. For example, when mixing a batch, the agitation time typically is determined by the amount to be mixed.

The batch recipe must also specify *connectivity requirements*. For example, if two substances are going to be sifted together in a premixer and then added to the batch proper in a reactor, the premixer chosen must be able to dump into the reactor where the batch is cooking. In many batch plants, it is too expensive to make all possible connections, so connectivity is an important factor in correct scheduling.

The Plant. The plant configuration often provides the most important limitations on productivity. We have already discussed the question of connectivity. The model of the plant to be scheduled must contain specifications of the units, their capacities, and the connections between them. In very large plants, it will also be important to specify the rate at which substances can be moved along connections. It is possible that the time a batch spends in various pipes will be the determining factor in the batch duration.

A very important feature of plant configuration is the amount of storage present in the plant. If a product cannot be stored or shipped immediately, it should not be produced! The plant model must specify how much tankage (or other storage) is available, what substances can be stored in what tanks, what the tanks are connected to, and so on.

Handling storage is a key requirement for batch scheduling. One reason that previous scheduling systems have not been popular in batch process industries[v] is that they have not successfully tackled storage management. In many batch processes, storage capacity is a key limiting factor; that is, if we make a batch of product, we must be certain that there is an appropriate tank (or other storage location) in which to put it. Likewise, we must postpone production of a particular substance until such time as we have sufficient stocks of all the raw materials. Previously, scheduling products for batch manufacturing have been job-shop schedulers for discrete manufacturing that have been minimally extended. They have not been able to take into account the continuous aspects of the batch scheduling problem.

Changeovers. One aspect of batch manufacturing not handled by previous scheduling systems was the issue of changeovers between activities. Often activities in a batch process will leave a piece of equipment in a particular state, and this state will have a profound effect on the next activity to use that equipment. To this end, we model some activities as having state requirements and as having a duration that are a function of the immediately preceding state of the equipment.

Miscellaneous Constraints. Miscellaneous constraints are used to capture other features of the scheduling problem. Two particularly important kinds of constraints are nonlocal limitations and user constraints. Nonlocal limitations are constraints concerning interacting activities. One particularly important class of constraints is setup and cleaning times. Another important class of constraints is labor supply. Often the pool of plant labor is not tied to particular units, but simply specifies a plantwide limitation on task completion.

User constraints are important to capture features of the problem that are not amenable to explicit modeling. For example, rather than specifying a complex model of preferences, the plant manager may simply wish to specify that batch A should be done before batch B, because the manager knows that the customer who has requested batch A needs it more urgently than the customer for batch B.

4.7.3.2 Solution

A schedule is a solution to the above problem. To be valid, a schedule must contain the following elements:

- Assignment of resources to each activity;

[v] Or the continuous process industries, for that matter.

- A partial order over the set of activities in the schedule, such that the resulting schedule has no possible resource contentions.

The partial order is constructed by making a set of pairwise ordering decisions on activities in the schedule. These decisions are made to avoid resource contention.

4.8 Constraint-Envelope Scheduling

Over the past years, the Automated Reasoning group at HTC has developed and implemented the techniques required to solve large, complex scheduling problems. Solving a scheduling problem does not mean simply implementing an algorithm for solving a particular constraint satisfaction or constrained optimization problem. For many organizations, constructing schedules is an extended, iterative process that may involve negotiation among various agents, scheduling choices made for reasons not easily implemented in an automatic scheduler, and last-minute changes when events do not go as expected. Any attempt to provide a useful scheduler must consider the process by which a schedule is constructed and used in the organization.

In our approach, which we call *constraint-envelope scheduling*, schedules are constructed by a process of "iterative refinement" in which scheduling decisions correspond to constraining an activity with respect to either another activity or to some timeline. The schedule becomes more detailed as activities and constraints are added. Undoing a scheduling decision means removing a constraint, not removing an activity from a specified place on the timeline.

CES is a *least-commitment* approach. We do not assign a set of activities to places on a timeline, assigning each activity a start and end point. Rather, we collect sets of activities and constrain them only as needed. Constraints may be in terms of relations between activities (e.g., the dumping phase of a reactor activity must follow the heating phase) or relative to metric time (the heating phase takes 2 hours, or the batch must be done by the close of business on Tuesday). Additional constraints are automatically added only as needed to resolve conflicts over resources. For example, two activities that use the same reactor in a plant must be ordered with respect to one another. On the other hand, if these two activities used *different* reactors, their ordering would not have to be determined.

The least-commitment nature of our schedules is an important advantage when it comes to rescheduling. If an event arises that makes a resource unavailable, or an ongoing activity takes longer than expected, the effect on the schedule is minimized. First, only those activities related by a chain of constraints to the activities explicitly moved will be affected. Second, if the set of constraints in the schedule is consistent with the new event, the projected effect of the schedule can be updated efficiently without any rescheduling at all.

Our schedulers distinguish between schedule *modification* (or schedule update) and rescheduling. For example, if one step of a complex process takes longer than expected, but the process (and the other processes that follow it) can still be completed before their deadlines, then all that needs happen is to update the estimated start and completion times—a task that can be accomplished very quickly. No reordering or reassignment of activities need take place. Only if some constraint is violated (e.g., an activity cannot be completed by its deadline) need we actually reschedule and change the scheduling decisions previously made.

The assumptions underlying constraint envelope scheduling are as follows:

1. Explicitly modeling the constraints resulting from specific scheduling decisions makes the schedule easier to construct and modify.
2. Representing only those relationships required by the current set of constraints (the decisions made so far) provides a more useful picture of the current state of the scheduling effort.

The main consequence of this approach is that the scheduler does not manipulate totally ordered timelines of activities and resource utilization. Instead, the evolving schedule consists of a partially ordered set of activities becoming increasingly ordered as additional constraints are added (or less so, as those decisions are rescinded). This basic approach is not unique to Honeywell. What *is* unique to our approach is the support provided for reasoning about partially specified schedules and the techniques we use for searching through the resulting space.

Although providing increased flexibility (through delaying commitment), the explicit representation of partially ordered activities in the time map makes reasoning about resource usage and other state changes more complicated. It is no longer possible to construct a single timeline representing, for example, changing resource availability over time. Instead, the system computes *bounds* on the system's behavior.

Despite the approximate nature of this reasoning, we are ahead of the game: where the least-commitment approach to scheduling can at least provide approximate answers in support of scheduling decisions (for example, what order activities should occur in), timeline schedulers make the same decisions arbitrarily—putting an activity on the timeline is a stronger commitment than constraining it to occur, say, between two other activities or within a given time window.

Our successful application of CES relies heavily on two supporting facilities. The first is a set of sophisticated search techniques loosely based on an extension to Ginsberg's dynamic backtracking.[vi] Dynamic backtracking is a smart search strategy in which the stack can be rearranged during search so as to focus attention on the relevant parts of the problem. This rearrangement is directed and controlled through the application of a set of specialized no-goods called *eliminations*, based on the set of assigned variables. These eliminations are explanations—in terms of other decisions—for why a particular decision cannot be made. For example, in a batch scheduling problem, one might not be able to process a batch B1 on a particular reactor (reactor A) because another job (B2) is using the reactor. So the reason (elimination explanation) we can't assign reactor A to B1 is our earlier choice to assign it to B2. What that means is that we must either choose a different reactor for this job, relax its deadline, or move the other job.

Our extensions to dynamic backtracking include the ability to return inconsistent sets of variables upon failure and to attempt assignments that are not known to be consistent.

[vi] Ginsberg, *op. cit.*

These techniques provide us with the ability to tackle large constraint satisfaction problems efficiently and to provide information about sets of inconsistent constraints. These constraints provide an explicit record of previous decisions (deadlines, activity orderings, etc.), allowing us to provide guidance to users in reformulating problems when, for example, a deadline is too tight or resources are oversubscribed.

We also make use of a highly optimized temporal constraint engine. Determining the impact of a scheduling decision on a set of temporal constraints is the activity that is the "inner loop" of our scheduling approach. Our search engines must be able to explore the consequences of many different scheduling decisions. This is made possible by the effective exploitation of an extremely efficient temporal constraint manager called the *Interval Constraint Engine* (ICE), which is tuned to provide efficient support for a dynamically changing graph of temporal constraints. The underlying representation manipulated by ICE is a graph of time points t and constraints of the form $t_2 - t_1 \leq d$. The inference supported by ICE involves proving or disproving possible relations between points, given the current set of constraints. For example, in adding an ordering between activities to resolve a resource conflict, ICE would be consulted to determine which of the possible orderings was feasible given the current state of the schedule.

ICE is a difficult level at which to encode a scheduling problem, because it simply talks about time points and the relationships between them. Accordingly, we support our schedulers with an activity and task model that captures common features of a wide variety of scheduling problems. These different problems had widely differing kinds of significant activities. However, in all of them, a scheduler needs to be able to do similar things to activities—put one before another, choose what resources to assign to meet the requirements of an activity, and so on. Accordingly, we developed a library of objects and methods encapsulating the common features.

The differences between scheduling problems cannot be wished away, however. Building a scheduler for, NASA EOS[vii] data archiving operations for example, differs substantially from building one for batch manufacturing or avionics software. These different domains have widely differing kinds of significant activities, so each scheduling problem requires us to design new activities and methods that inherit from those in the common task and resource model.

The interaction between the search engine (the dynamic backtracker) and these domain-specific activities is mediated by *flexible constraint objects* (FCOs). We developed FCOs so that we would not have to reengineer the search engine for each new scheduling problem. The FCOs capture the common features in interactions between the search engine and the domain model. Essentially, a search engine makes decisions or assigns variables. These decisions have to be translated into effects on the domain model (like putting one activity before another). The FCOs are objects that encapsulate the transactions between the problem solver and the domain model. Many of these can be reused from scheduler to scheduler—for example, the FCO that handles the decision to put activity A before activity B, or vice versa. Others have to be built specifically for a

[vii] Earth Observing System.

given scheduling problem—for example, the FCO that handles the effects of assigning a given ground station to handling transmissions from a particular satellite.

4.9 System Requirements

In this section, we discuss how the batch scheduling program must interact with other elements of enterprise management in a manufacturing site. We start by discussing how human schedulers should work with the scheduling program. On the basis of our experience with manufacturing enterprises and with several previous scheduling systems, we argue for a *mixed-initiative* approach in which computer system and human scheduler work together to solve the scheduling problem. We discuss this in Section 4.10.

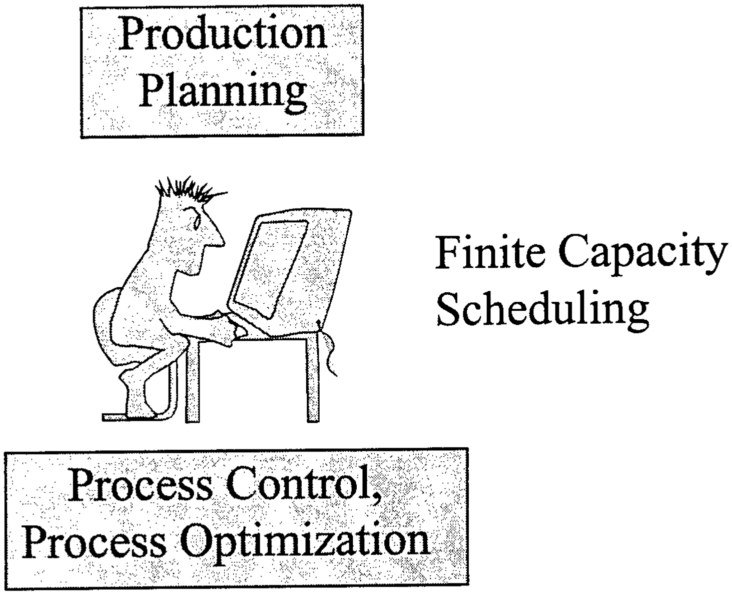

Production Planning
Finite Capacity Scheduling
Process Control, Process Optimization

Figure 24 Manufacturing Communication Gap

Automated scheduling systems provide an opportunity to solve a key problem in manufacturing integration: overcoming the gap between business management (the "bean counters") and plant operations personnel (see **Figure 24 Manufacturing Communication Gap**). Business-oriented personnel in the batch (and discrete) manufacturing industry organize operations using production planning systems, typically based on some flavor of materials[viii] resource planning (MRP or MRP-II) system. Plant personnel, on the other hand, work in terms of process control and optimization, typically using some form of distributed control system (DCS). The world view of these two systems diverges widely.

One critical difference between production planning and process control models is that the MRP system models simply consider limits on the rate at which various products can be made, and they consider these rates *independently*. For example, if there are two products, *a* and *b*, the MRP system will contain information about how much of each can

[viii] Sometimes written as "manufacturing" rather than "materials."

be made in a day. However, the MRP system will *not* specify whether or not *a* and *b* can be made simultaneously. The person using the MRP system will not know whether or not *a* and *b* require the same piece of production equipment and may end up generating schedules that are not feasible when the plant's *finite capacity* is taken into account.

In the design of MRP systems, the above gap is supposed to be filled by finite-capacity scheduling. However, there are few very good finite-capacity scheduling systems available at all, and no good ones for batch manufacturing; finite-capacity scheduling for discrete manufacturing is much better understood than capacity scheduling for batch. The upshot is that most batch manufacturing enterprises are left in the unhappy situation shown in **Figure 24 Manufacturing Communication Gap**.

The communications gap is not simply a problem of communicating top down. Bottom-up communication is also affected. It is very difficult to determine the effect on business concerns of events at the DCS level. For example, at this level, one typically only has aggregate quantities of products and raw materials, rather than customer orders. Often it is even difficult to identify what events have happened—all one has is measurements of particular *process variables* such as vessel temperatures, etc.

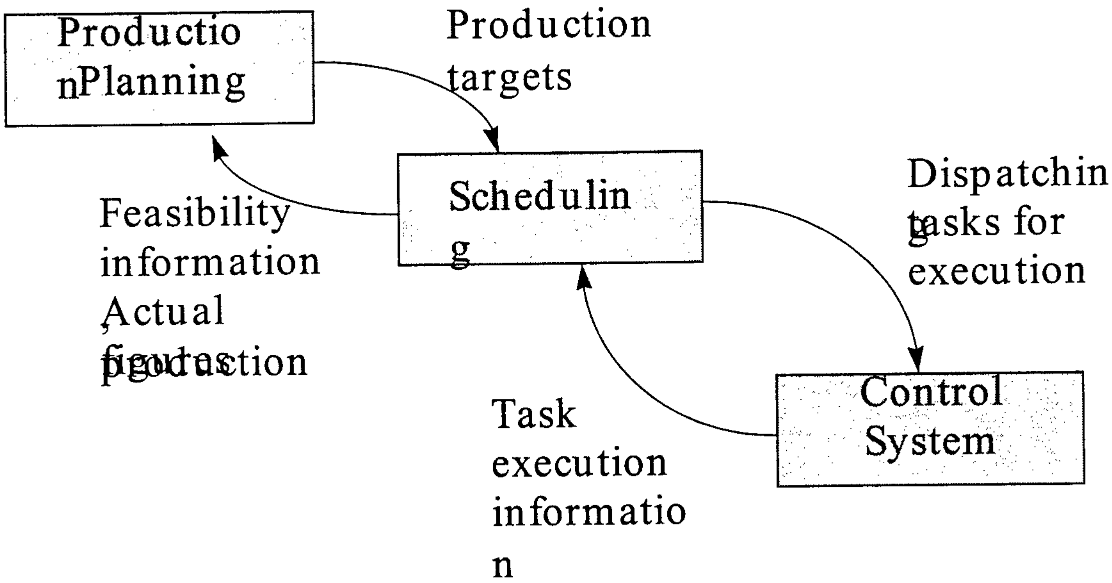


Figure 25 Bridging the gap between business and operations.

Schedulers must bridge the gap between operations and business concerns, as Figure 14 shows. However, most human schedulers have no tools to help them in this task. The information generated by production planning and process control systems is not aimed at their concerns. Our scheduling technology bridges this gap by communicating directly with production planning and distributed control systems, as shown in Figure 25. To do this, the scheduling system must be able to read data from DCS and MRP systems and also produce information that those systems can import.

4.10 Mixed-Initiative Interaction

Any scheduling system for batch manufacturing must support mixed-initiative interaction. What this means is that the automated system and the human user must be able to work *together* to construct a schedule. Successfully supporting plant management in batch manufacturing draws on both human expertise and the patient exploration of many scenarios, of which modern, high-powered workstations are capable, especially when aided by clever algorithms.

Human expertise is needed because of limitations in the models used by automated scheduling systems. For example, consider a case where a plant is being used at full capacity and then a new order is received from customer *A*. Customer A is a very important customer to the enterprise and needs this order completed immediately, to satisfy one of its own key customers. We do not want the plant manager to have to devise a complex objective function or set of rules to ensure that customer A's job will be given priority. The plant manager should simply be able to insert the new order into the schedule at an appropriate point, and the scheduling system should appropriately adapt the schedule.

That example concerned the setting of objectives. Plant personnel also often have experience that is difficult to capture. For example, consider the case of a food oil plant. The plant engineers may simply know that, because of hot, humid weather, it will take longer than normal to winterize a batch of oil. They are not likely to have an in-depth physical or mathematical model from which this can be predicted. Despite this, their knowledge is reliable and should be exploited. It should be possible for a plant engineer to examine the schedule, say "this operation will not be completed in 8 hours; I think it will probably take 12," and simply make the required edit to the schedule (e.g., by stretching a ribbon on a Gantt chart display). Again, the automatic scheduling program should respond appropriately.

So far, we have talked only about why the automatic scheduling system needs to respond to human knowledge. What does the automatic scheduling system bring to the table? The scheduling system supplies two essential capabilities: *projection* (or simulation) and *search*.

Projection—To schedule any process, one must be able to make scheduling decisions (such as "start batch *a* before batch *b*" or "start batch *a* at 12:00 tomorrow") *and* determine what effect these decisions will have. In complicated processes, this projection may be very difficult, since a single, early decision may have ramifications throughout the period of the schedule. An automatic system, armed with a process model, can do better than humans at tracking out all the ramifications of scheduling decisions (provided the model it is given is accurate).

In many applications, providing model-based projection alone is sufficient. This is not the case in batch manufacturing. In even relatively modest batch plants, an immense number of scenarios must be explored in making a schedule. We know of many cases where a sophisticated scheduler, trying to solve his or her scheduling problem, has developed a spreadsheet that is capable of projecting the effects of scheduling decisions, yet the scheduling problem remains too difficult. The problem is that human beings are ill-suited to exploring a large space of possible scheduling decisions *systematically*. This *search* task is better left to a computer.

However, search alone is not sufficient. The scheduling system must be able to effectively communicate the results of its search to users. For example, any effective scheduling system for a moderately complex domain needs to support the user in constructing an initial schedule, modifying an existing schedule, and detecting and characterizing unfeasibility. The last of these functions is not universally recognized as crucial, but we have found it to be of great importance. The initial stages of solving a scheduling problem can involve an extended period in which those responsible for generating the initial schedule requirements must be persuaded to modify those requirements to achieve consistency.

4.11 Interaction with Control System

To meet the needs of enterprise integration, the scheduling system must be able to do two things:

1. Generate instructions that the control system, with its human operators, can execute.
2. Track what actually happens on the plant floor in order to:
   a. Update its own model, so that the schedule continues to be an accurate picture of plant events and capabilities;
   b. Provide feedback to the business planners about the successful completion of their plans (or any disruptions).

4.12 Sending instructions to the DCS

In the current state of the art, automatic generation of instructions for execution is difficult because batch control systems are not standardized. Accordingly, there is no standard format in which these orders can be generated, nor is there even a firm set of concepts common to all batch control systems. The ISA SP88 standard aims to fill this gap, but it is not yet completed. For purposes of this project, we have constructed a batch control system simulator based on our knowledge of currently available batch control systems and features likely to be present in future batch control systems. It made the most sense for us to use a simulated system to develop a demonstration system while productization plans are firmed up. The interface with the simulated system was based loosely on our knowledge of Honeywell batch control products like Batch Supervisor PC (BSPC)™ and TotalPlant Batch™.

We did not assume a very sophisticated interface, but simply the ability to trigger particular recipes with particular scale factors. Often batch recipes are encoded in terms of a standard batch size, in much the same way cooking recipes give a standard amount, say, "24 medium-sized cookies." Like a chef with an especially large or small group of diners, the batch plant operator will adjust the scale of a recipe within the limits of the plant and the recipe—for example, one can't brew more beer in one batch than the largest vessel will hold, and there's some minimum amount below which the batch will not be large enough to ferment correctly.

4.13 Retrieving information from the DCS

Typically, distributed control systems provide information about significant events by producing an *event stream,* or *history stream*. The event stream will be made up of time-stamped event tokens. In a regulatory control system, the plant history is likely to be made up simply of measurements of significant elements of plant state (usually referred to as *points*). In more sophisticated batch control systems, there will also be event tokens in the history stream for significant events in the execution of batch recipes, for example, the beginning or end of a unit recipe. Many distributed control systems provide a plant history database, such as Honeywell's Uniformance™. A plant history database relieves parties interested in significant events from the necessity of watching and parsing the event stream. Instead, they can simply query the database in a standard relational format, and some plant history databases (e.g., Honeywell's Uniformance with InterPlant™ Operation Events) will issue notifications when significant events occur.

In our demonstration system, we have built a plant DCS simulator that generates a stream of time-stamped recipe (and phase) begin and end events. The scheduling system reads these events and, based on them, updates its schedule with actual begin and end times. The effects of these can ripple out. For example, if the schedule dictated that batch *B* must start between 9:00 and 10:00 and the scheduler learns that it has actually begun at 9:30, the scheduler will update its estimates of when other activities will occur. Activities that must wait for *B* to end—either because they must work on what *B* produces or because they must wait for *B* to finish to get access to some piece of equipment—will have their time intervals updated.

As the schedule in the scheduling tool is updated, the HBS will check the estimates of completion time against the orders it has received from the production planning system (see Section 4.11) and generate updates to the production planning system, so that business personnel can determine when customer orders will be fulfilled and when more raw materials will be needed.

4.14 Interaction with Production Planning

MRP-based production planning works in terms of inventory to meet orders (both known customer orders and expected orders, projected from past demand). Interacting with MRP planning, the scheduler must do two things:

1. Provide feedback to the planner about the feasibility of his/her production plans and
2. Update the planner's inventory information based on actual plant performance.

4.15 Closing the planning loop

As mentioned earlier, MRP systems do not take into account the finite capacity of actual plants. Therefore, it is possible for them to generate production plans that are not feasible. The HBS provides feedback to the production planner's MRP system, so that production plans can be adjusted to fit the realities of plants.

To understand the role of finite-capacity scheduling, it will help to know a little about how MRP systems work. MRP systems generate production orders for finished products based on projected demand. For more JIT-style enterprises, production orders for finished products will be generated directly from customer orders.

The MRP system will determine, from the finished products to be made, what intermediate products need to be made (e.g., in a margarine plant, before one can make margarine, one must produce deodorized oil from some raw vegetable oil), and from the intermediate products, what raw materials must be procured and when they must arrive.

The MRP system generates a set of planned orders in this process. Some of these orders become *firm* planned orders.

Interacting with the MRP system, the scheduler accepts a set of planned final product orders (it can actually handle intermediate products as well, along the same lines). These come in with release times and deadlines. The scheduler will either successfully schedule all of the orders or indicate an infeasibility. In the event of scheduling all orders, the scheduler will return a more accurate picture of when production will take place. This can be used to update the MRP system's planned production orders to firm planned orders. From this new estimate, the MRP system will be able to compute more accurate estimates on when raw materials (and intermediate materials) will be needed.

Sometimes the set of orders that the production planner requests will not all be satisfiable. In this case, the scheduler will recognize the infeasibility and identify a subset of the requests that create an inconsistency. For example, if the production planner requests three batches of yogurt on Wednesday afternoon, the scheduler may report that these cannot all be done, say, if there is only one vessel available for fermentation. The BMS will *not* simply say no. It will distinguish the set of orders involved in the infeasibility from other activities (e.g., other yogurt batches needed later in the week and orders for other products that do not require a fermentation vessel). This feedback will give the production planner the opportunity to revise the orders, either by relaxing deadlines or release times or by canceling orders. This process of planning and feasibility testing will be an iterative one, continuing until a schedule agreeable to both parties can be devised.

4.16 Updating plans based on actual plant events

The interaction with production planning will not stop when an initial schedule is derived. That would only work if production were completely predictable and reliable, which no real process is. In fact, processes get delayed, equipment breaks, customers cancel orders, and so on.

Process delays and equipment breakdowns will be noticed by the BMS, either from its connection to the plant control system (see Section 4.11) or from notification by plant personnel (e.g., in cases where an expert can predict, based on past experience, that some activity will take more or less time than expected). The BMS will then roll these events into its schedule, determine whether any will prevent the planner's objectives from being met, and, if so, notify the planner.

Schedule upsets can occur from the top down as well as the bottom up: for example, urgent new orders can arrive, orders may be canceled by customers, or necessary raw materials may fail to arrive. In that case, updates will have to be generated from the production planning/inventory system to the BMS. The BMS will pick these up, and the schedule can be updated, possibly involving a discussion with the production planners along the lines outlined in the previous section.

System Architecture

During this contract, HTC has implemented a Batch Manufacturing Scheduler system. To demonstrate the integration potential of the BMS, HTC, with help from Template Software, has also constructed a simple enterprise integration simulator. The architecture of this integrated manufacturing logistics demonstration is shown in

Figure 26 Enterprise Integration Architecture

. The components built by Template Software are illustrated in gray, and those built by HTC are shown in green. The Template components, and communication between them and the scheduler, were built using Template Software's Workflow Template.

The scheduler itself is decomposed according to a functional, object-oriented structure. Major components of the scheduler are illustrated in **Figure 27 Scheduler System Architecture**. As far as the core functionality of scheduling is concerned, the scheduler operates by transactions between a *problem solver* (or search engine) and a *domain model.* The domain model is the formulation of the scheduling problem. It is the search engine's job to explore possible alternative decisions and the domain model's job to determine whether a decision is good or not. Communication between the domain model and the search engine is encapsulated in a set of flexible constraint objects, or FCOS. This intermediate layer allows us to have generic search engines, applicable to many problems; the FCOS translate decisions made by the search engine into changes to the domain model.

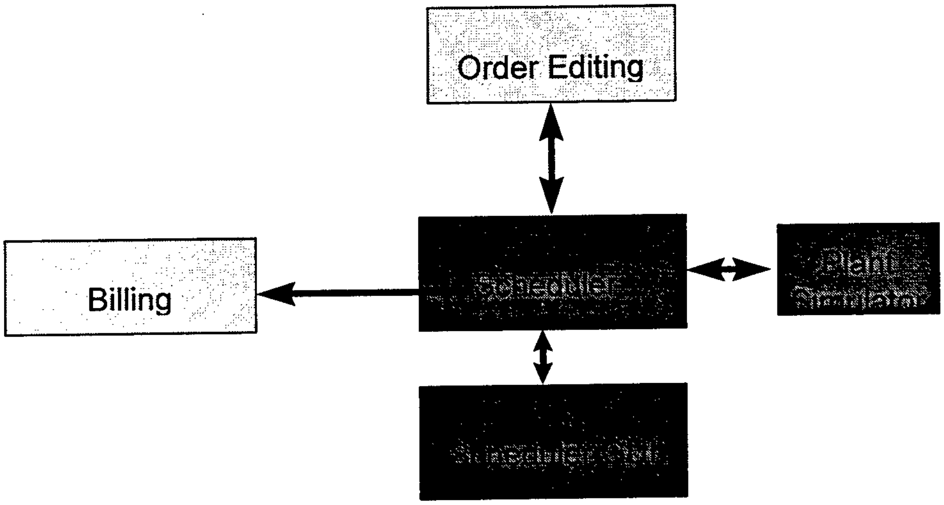


Figure 26 Enterprise Integration Architecture

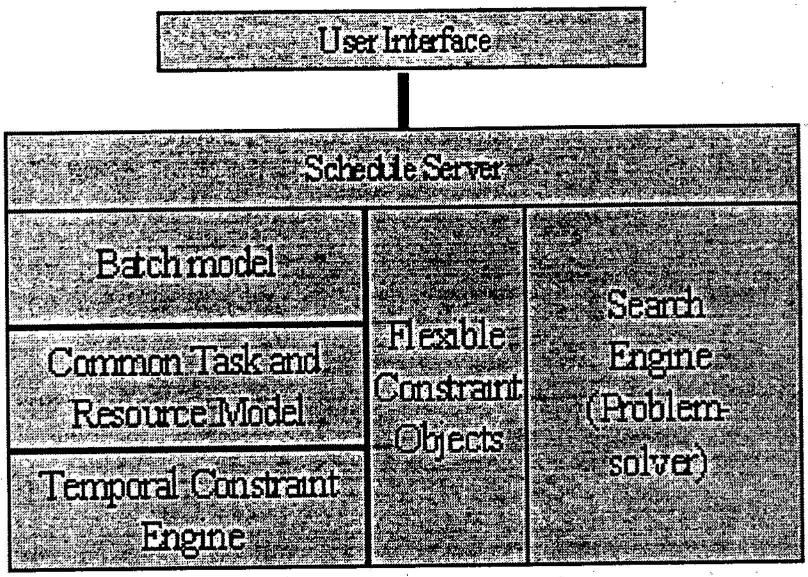

User Interface
Schedule Server
Batch model
Common Task and Resource Model
Temporal Constraint Engine
Flexible Constraint Objects
Search Engine (Problem-solver)

Figure 27 Scheduler System Architecture

The domain model as a whole is decomposed into three components: the domain model proper, the common task and resource model, and the temporal constraint engine. The temporal constraint engine (also referred to as the *Interval Constraint Engine,* or ICE) acts as a database of time points and constraints between those points. For example, when we decide to schedule activity A before activity B, we can do this by constraining the end point of activity A to be before (less than or equal to) the start point of activity B. ICE will record and propagate the effects of such constraints.

As our example above illustrated, when solving scheduling problems, we do not typically think about time points. Rather, we think in terms of *activities* and the relationships between them. In our example, we think of scheduling activity A before activity B, not of a relationship between time points. The *Common Task and Resource (CTR) Model* captures an activity-based model of schedule construction. It provides a layer of abstraction permitting us to talk about ordering activities rather than points. Typically, we need to order activities because they must use resources (e.g., plant equipment), and there is some limit on how much of a given resource can be used at any time. For example, if two activities need to use the same reactor, they can't both occur at the same time. Since reasoning about resources is central to scheduling, the CTR model provides attributes and methods to represent and reason about the resource requirements of activities.

The CTR model includes objects and methods that we have found useful across a wide variety of scheduling problems. However, we have always found that individual scheduling problems require their own special facilities. Accordingly, we complete the domain model by adding special objects and methods for the particular domain in which we are working (in this case, batch manufacturing).

The final two components are the *schedule server* and the *graphical user interface* (GUI). The BMS is built using a client-server architecture in which the scheduler as a whole offers services. The most common way to take advantage of these services is to interact with the schedule server through the GUI. However, in an integrated scheduling application, there are other interactions with the scheduler. For example, the scheduler must watch the *plant history stream* to observe events on the plant floor. The scheduler can then update its forecast of future events based not only on the user's plan, but also on what has happened in the execution of the plan The client-server architecture also simplifies our job of providing a scheduling system that multiple users can interact with concurrently. Multiple access is important because, in most organizations, responsibility for planning operations is shared among multiple stakeholders.

5. Domain Models

Following the principles of object-oriented design, the design of our BMS centers around a model of its domain, batch manufacturing. The batch object model we have developed is a layered one. The entities in the batch object model are all specializations of a common activity and task model (CATM).

5.1 Activity and Task Model

The Common Activity and Task Model (CATM) contains the object classes we have found useful in most, if not all, of our scheduling applications. The most important elements of the CATM are:

1. Activities,
2. Resources, and
3. Resource requirements.

5.2 Activities

A central purpose of the activity object class is to simplify programmers' interaction with the *interval constraint engine* (see Section 4.3). The ICE is able to track bounds on significant points in time and acts as a database and consistency checker in the scheduling process (see Section 6.2). However, it is often inconvenient to work in terms of simple time points. Rather, we often wish to work at the level of activities (e.g., "put activity *A* after activity *B*.") The activity object permits us to work at this level.

The activity class, in combination with the resource requirement class, also helps us track the use of resources over time, since resources are typically used by activities. We will discuss this later in this report.

5.3 Resources

In all but the simplest scheduling problems, the key issue is determining how to allocate resources to activities and how to spread those activities over time. In general, the need to sequence activities arises because they require resources. For example, in a simple job-shop scheduling problem, a number of jobs must be put through a number of machines. It is only because these jobs must compete for the machines that they cannot simply be done simultaneously.

Accordingly, resources are a central component of our CATM. Resources come in different classes, depending on the extent to which they can be shared. There are unary resources that can have only a single user at a time, like the machines in a job-shop problem. There are also metric resources (pools of indistinguishable resources). For example, a manufacturing plant with a steam heat system may only be able to heat some *n* reactors at a time. We could model this as a pool of heat of size *n*. Finally, there are consumable resources. These are things like tank inventory that are consumed by shipments, as opposed to simply being employed for a time, like the heat in the previous example.

5.4 Resource Requirements

Resource requirements are the final piece of specifying a scheduling problem. As stated earlier, it is only because of finite resources that scheduling problems arise. Resource requirements provide the way to map from activities to resources. Resource requirements are objects that can be attached to activities and that specify the resources the corresponding activity needs. Then, to solve the scheduling problem, we can identify resource contentions (sets of activities that *might* overlap and *might* overuse the resources). By scheduling those activities apart from each other, we solve the scheduling problem.

5.5 Batch Object Model

The classes in the batch object model are all specializations of CATM classes. The key extension we needed to make for batch manufacturing was a set of special hierarchical activities to capture batches made according to recipes. These hierarchical recipes are based on the specification of batch recipe terminology given in the ISA SP88 standard.

Batch recipes are organized into three levels:

1. Batch recipe,
2. Unit recipe,
3. Phase.

A *batch recipe* specifies the full set of activities needed to make a batch of a particular substance. In complex, multistage batch plants, this will be made of multiple *unit recipes*. A unit recipe specifies the set of actions that must be taken in a particular unit as part of the process of making a batch. For example, in brewing beer, there might be three unit recipes: one for the kettle or reactor in which the cooking takes place, one for the fermentation tank, and one for the filtration plant. Unit recipes are composed of *phases*, that correspond to individual control programs. From a scheduling point of view, phases provide information about the *synchronization* of activities and about activities' *use of resources*. To return to our beer brewing example, it is the way the *pouring* phase at the end of the kettle unit recipe lines up with the *filling* phase of the fermentation unit recipe that allows us to correctly sequence the batch as a whole. Likewise, we may track the use of additional resources, such as process heat at the phase level—(e.g., the *heating* phase of the kettle unit recipe will have a resource requirement for some amount of process heat).

6. Flexible, Object Oriented Constraint Satisfaction

One technology HTC has brought to fruition through this program is flexible, object-oriented constraint satisfaction. We have used object-oriented methods to develop *loosely coupled* search engines and domain models. In the past, search engines and the variables they worked on have been tightly coupled. This hinders reuse and modularity of design. We have overcome this problem using *flexible constraint objects.*

Our development of FCOs has been pushed by two design criteria. First, we need to be able to employ multiple, alternative search techniques in solving our constraint problems. Second, we wish to distinguish the domain model from the search model.

We have found that different scheduling problems benefit from radically different search methods. For example, in the scheduling of the Boeing 777 avionics, we used Ginsberg's dynamic backtracking algorithm and found that it provided performance we could not achieve with chronological backtracking (approximately two orders of magnitude). On the other hand, when attempting to build a scheduler for satellite operations, we found that the dynamic backtracking method could substantially hinder the scheduler. We have found there is room for a wide variety of search techniques, varying particularly on the dimension of how much effort is expended in identifying backtrack points. Therefore, we believe it is essential to develop an interface between search problem and search engine that permits the introduction of alternative search engines without a dramatic rewrite of the overall scheduler.

A second separation we would like to enforce is between the model of the domain to be scheduled and the search problem itself. A search problem is a specification of a set of decisions to be made, but the domain model is a specification of the actual activities in the domain. It is important that we be able to develop the domain model without making undue commitments to the design of the search problem, because scheduling problems vary widely in their characteristics, and the effect these variations will have on the scheduling approach are difficult to determine in advance. Accordingly, we often find ourselves wanting to be able to model the domain separately and only afterward choose an appropriate search engine.

The way that we have arranged to meet these needs is to have, first of all, a domain model. The domain model makes use of the Common Task and Resource Model discussed earlier. We also have a search engine, or problem solver (or actually, a set of alternative search engines to apply). This search engine works on a Search Problem object. The key responsibility of the search problem is to supply a set of variable objects for the search engine. The search engine solves the scheduling problem by assigning values to these variables until all variables have been assigned and the set of variable assignments is consistent. To determine whether variable assignments are consistent, we must translate assignments of values to variables into scheduling decisions that affect activities and determine the effect of these decisions. In other words, we must translate between the language of assignments that the search engine operates in and the language of orderings of activities, assignments of resources to activities, that is the province of the domain model.

The translation between variable assignments and scheduling decisions is made by FCOs. These objects are attached to search variable objects. By communicating with the domain model, FCOs are able to answer key questions for the search engine, such as: "What values of a search variable can be assigned, consistent with assignments to other variables?" and "What assignments to other variables rule out assigning a value to this variable?" The FCOs also translate the assignments made by the search engine into scheduling decisions whose effects will be understandable by the user.

6.1 Search Engines and Search Problems

As stated earlier, a search engine is a procedure that solves assignment problems. An assignment problem is one of assigning values to a set of *variables* such that every variable has a value and that these value assignments obey the constraints between variables.

It is not a simple process to phrase a scheduling problem in such a way that a search engine can solve it. Consider three alternative ways to schedule a number of jobs on a single machine:

1. One could have a set of variables, one for the start time of each activity. The constraints between variables would be that the start variables must be assigned so that the activities will not overlap.
2. Alternatively (and this is an oversimplified description of the approach we take), one could create a different set of variables, one for each *pair* of activities, and for each pair of activities, choose which one of them comes first. Then the constraints must be such that the activities can actually be ordered in this way. For example, the constraints must ensure that we cannot put *A* after *B, B* after *C,* and *C* after *A*.
3. A final way to achieve the same task would be to have one variable for each activity. The variables would take on values from 1 to *n,* indicating the position of the corresponding task in the order. So if the variable for task *A* took on value 5, *A* would be the fifth task. *A*'s start time could be calculated from the start time of the tasks assigned indices 1, 2, 3, and 4.

The most effective way to pose the problem to a search engine will vary, depending on the characteristics of the problem.

Not only are there various ways of presenting a problem to a search engine, there are widely differing *search strategies* as well. A search engine is simply a program that carries out a search strategy. The simplest, most common search strategy is *chronological backtracking:* we go through a list of variables, assigning a value to each one in turn, until we come to a variable that cannot be assigned a value. At this point, we *backtrack* to the immediately preceding variable, unset it, try the next possible value for that variable, and then move forward again. On many problems, chronological backtracking is not sophisticated enough. We have experimented with several other search regimes, including: dynamic backtracking, backjumping, and dependency-directed backtracking. All of these spend varying amounts of effort (time and space) to cleverly choose where to backtrack.

Although search problems and search engines can vary, what the search engine does will nevertheless be the same: it searches for an acceptable assignment of variables. Essentially, the search engine must be able to:

1. Get a new variable to assign,
2. Check to see if it has successfully completed the problem,
3. Assign a value to a variable,
4. "Unassign" a value to a previously assigned variable.

Wherever we have a standard interface, we may introduce objects to capture the essence of this interface, hide information, and simplify code reuse. That is what we have done here, first by introducing a class of *Search Problem* objects.

Essentially, what a Search Problem object does is handle the presentation of variables to the search engine. That is, the search engine can request a new variable from the search problem (or, more generally, can request a set of remaining variables, from which it can attempt to choose the best to work on next), and can ask the search problem whether the it has been solved yet.

The Search Problem object(s) for a scheduler, then, capture the global aspects of framing the scheduling problem in terms appropriate to a search engine. However, Search Problems are not enough, because they do not capture the second two interactions in our list: they do not handle assignment and unassignment. These behaviors are handled by variable and flexible constraint objects.

6.2 Variables and Restrictions

Our initial attempts at object-oriented design had, in addition to search problems, only search variables. These search variables encapsulated two behaviors (and, hence, two "transactions" with the search engine): assignment of value and retraction (or "unassignment"). However, search problems and search variables alone turned out not to be sufficient, and we were led to introduce *flexible constraint objects (FCOs).*

The two behaviors that a variable must support—assignment and unassignment—are by no means simple ones. In particular, when asked to take on a value,[ix] a variable must do two critical things:

1. Update the domain model to reflect the variable assignment and
2. Determine whether or not the assignment request is valid and, if not, identify what other variables' settings make it invalid.

[ix] Taking on a value is what a variable does in response to an assignment request from the search engine.

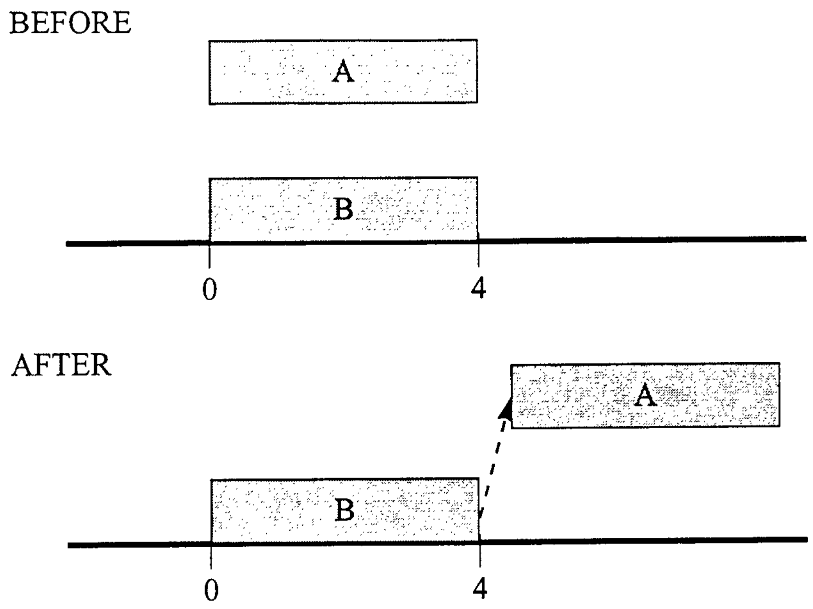


Figure 28 The effect of assigning a variable

Let us consider a variable assignment that corresponds to the decision to put activity *A* after activity *B*. When this assignment is made, the domain model must be updated accordingly. For example, in our framework, we add to the temporal constraint graph (see Section 4.3) an edge between the end point of activity *B* and the start point of activity *A*. We show this process in **Figure 28 The effect of assigning a variable**. Note that after the variable is assigned, the model is updated so that we can see that *A* must start after time 4.

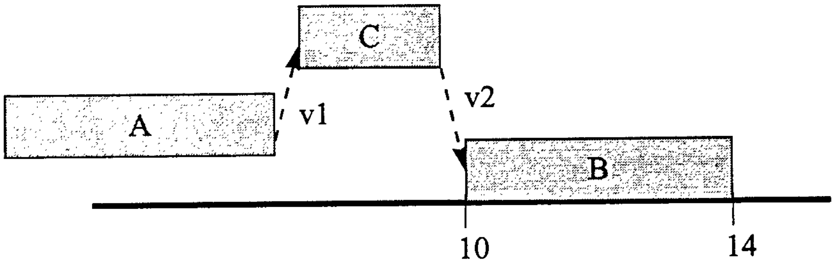


Figure 29 An inconsistent assignment

Now consider what happens when we try to assign the same variable *inconsistently*. This scenario is shown in **Figure 29 An inconsistent assignment**. In this situation, we have previously assigned variable *v1,* putting *A* before *C* and *v2* and putting *C* before *B*. So we cannot put *A* after *B;* it is simply impossible. It is not enough that we simply report that the assignment fails. Instead, the variable must also indicate *why* the assignment has failed—in this case, because of *v1 's* and *v2*'s assignments. This explanation can be used by the search engine to backtrack to one of those earlier variables in the case of a failure. It can also be presented to the user to explain a scheduling failure.

However, it turns out that the search variables are not the appropriate objects to encapsulate the behaviors we have just discussed. The search variables *do* encapsulate the behaviors of assignment and unassignment, but we need additional objects to capture the process of *putting these assignments into effect.*

These additional objects, which we call *flexible constraint objects,* are associated with variables and capture the interactions between variables. They are best modeled as separate objects because they capture interactions between other objects. For example, consider three variables:

1. A variable that represents the choice of workstation for activity *A*;
2. A variable that represents the choice of workstation for activity *B*; and
3. A variable that represents ordering between *A* and *B*: the decision to put *A* before *B,* put *B* before *A,* or allow them to occur in any position.

Now, when variables 1 and 2 are bound to the same workstation (i.e., both *A* and *B* must be done on the same machine), *A* and *B* cannot occur simultaneously; that is, variable 3 must be set to either "*A* before *B*" or "*B* before *A*." Conversely, if variable 3 is be set to either "*A* before *B*" or "*B* before *A,*" variables 1 and 2 may be set to the same value, otherwise they must be set to different values.

Conceptually, the behaviors above are all one single constraint, and representing them three different times, in three different places (variables 1, 2, and 3), is poor design. Instead, in cases like this we design a single FCO that can be attached to each of the search variables. These FCOs are invoked when a variable is assigned or unassigned, and it is the responsibility of the FCO to put assignments into effect. The advantage s that search variables are completely generic and portable between applications.

7. Development Process

We can identify four phases to our development process under the OTRSD TRP. The first centers around the previously existing BMS prototype, written in Common Lisp. In this phase, we used that initial prototype as a reference point in designing the commercializable BMS and experimented with alterations to the design, often by applying these alterations to the running prototype before incorporating them in our product design. The second phase centers around the new version of the BMS, gradually leading to the abandonment of the Common Lisp prototype. The most significant change of direction in this phase was to build the core functions of the BMS in C++ rather than in Java. The third phase, overlapping with the second somewhat, consists of working with Template Software to construct the Enterprise Integration Demonstration. Finally, during an extension of the contract under a modification to our original SOW, we have been investigating how the technology developed in this program may be applied to continuous manufacturing, in particular, petroleum refining operations.

7.1 Initial prototype

As discussed earlier, we began this program with a working prototype BMS written in Common Lisp. This prototype provided most, though not all, of the desired functions but was not acceptable to Honeywell divisions as a basis for technology transfer. Some screen shots of the initial prototypes are shown in Section **Error! Reference source not found.**.

Our first task, then, was to design the product version of the BMS. This was done initially using the Rational Rose™ tool (see Section 8.2). At the same time we were designing the new version of the BMS, we were familiarizing ourselves with the key concepts of the Consortium technology: SNAP™ and CORBA.

In keeping with our philosophy of iterative development, we kept the initial prototype in working order in the early stages of this project and used it to experiment with new features and development strategies. For example, our earliest experiments with distributed systems structure used the Common Lisp prototype, first with sockets and then with CORBA (through Xerox PARC's ILU; see Section 8.4.2).

7.2 Iterative Development: BMS Reimplementation

The division between the phase of working with the Common Lisp prototype and work on the BMS reimplementation is not an absolute break. We had begun work on reimplementing the lowest levels of functionality as early as the second quarter of this project. As mentioned earlier, this phase divides into two subphases. In the first subphase, we were developing a C++ implementation of the BMS. Although this work resulted in an initial working version, there was growing disappointment about features of the C++ language. When Java became widely available toward the end of 1996, we became increasingly excited about its possibilities. This culminated in the eighth quarter of the contract, when we translated the C++ version of the scheduler into Java, which has been the basis of all our follow-on work.

7.3 C++ implementation

Essentially, the C++ implementation phase of the project ran from the second through the seventh quarters (from approximately June 1995 until December 1996). We took the C++ version to a relatively complete BMS system (though not integrated with the other aspects of the simulated batch enterprise). By that time, however, we were concerned that the characteristics of C++ made this new version brittle, difficult to maintain, and a poor basis for code reuse (for a discussion of these problems, see Section 8.3.3). These problems led us to move to Java, as discussed in the following section.

In the second quarter of the project, we had complete C++ versions of two core components of constraint-envelope scheduling: the Interval Constraint Engine and the Dynamic Backtracking Search Engine. In this quarter, we also worked with Template Software to suggest enhancements to their SNAP product, particularly in the area of user interfaces, to enable it to deal with complex scheduling systems. At this point, we were still concurrently developing the overall BMS design and were experimenting with new functions in the original prototype.

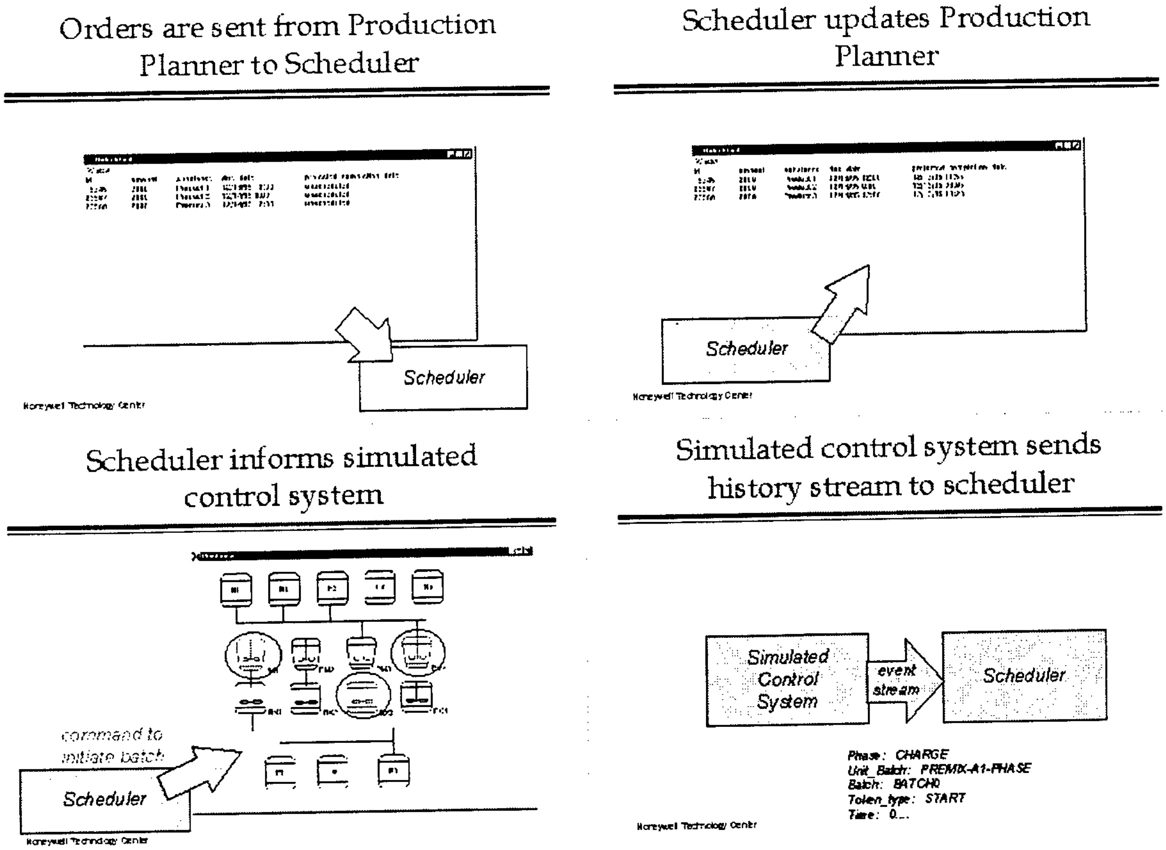
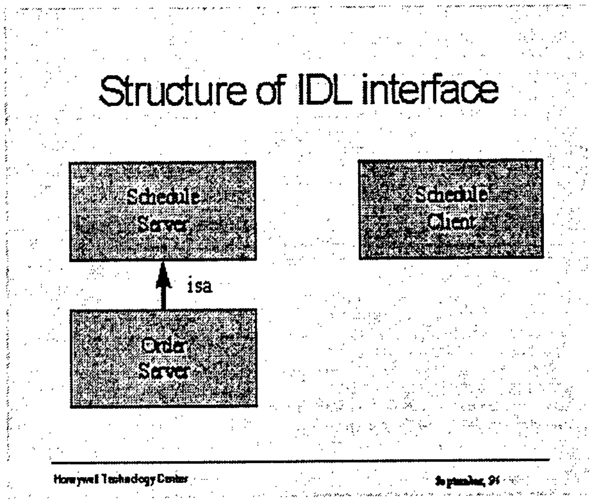

Orders are sent from Production Planner to Scheduler
Scheduler updates Production Planner
Scheduler
Honeywell Technology Center
Scheduler
Honeywell Technology Center
Scheduler informs simulated control system
Simulated control system sends history stream to scheduler
command to initiate batch
Scheduler
Honeywell Technology Center
Simulated Control System
event stream
Scheduler
Phase: CHARGE
Unit_Batch: PREMIX-A1-PHASE
Batch: BATCH0
Token_type: START
Time: 0...
Honeywell Technology Center

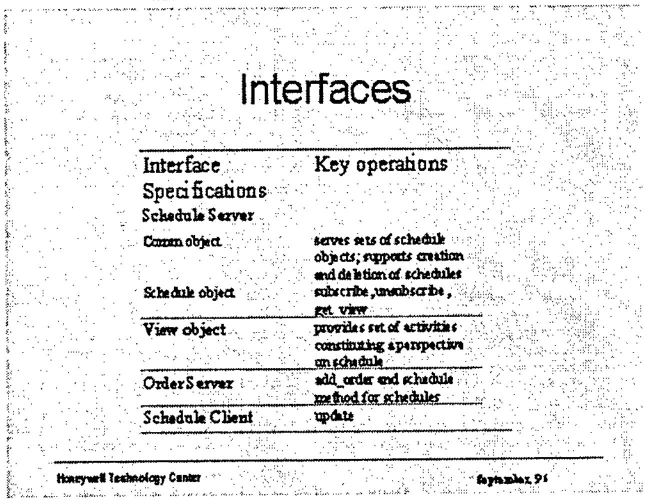
Figure 30 Original Design of Enterprise Integration Demonstration

In the third quarter, we focused on the design of the overall batch enterprise management system, sketching out the interactions discussed above in Section 4. Our presentation to the Consortium quarterly review meeting showed the basic structure in place, and this structure remained the same throughout the rest of the contract with minor changes (see **Figure 30 Original Design of Enterprise Integration Demonstration**). In this quarter, work on the implementation continued, but at a slower pace than hoped, due to a shortage of labor that was to plague us for several quarters; later in the program we made up lost time.

In the fourth quarter, we demonstrated the integration architecture for the first time. Unfortunately, because our development of the enterprise architecture ran ahead of development of a communications infrastructure, we had to simulate CORBA communication through raw socket connections, which made the demonstration very expensive to do. Furthermore, although the final version of the production planning component was to be written in SNAP, we had to build a mock-up ourselves in C++. This was regrettable but unavoidable. We felt it was important to keep to our iterative prototyping process of having a working system available at all times.

The distribution design and demonstration in the third and fourth quarters led to a more formal, CORBA-based distribution design that we presented in the quarterly meeting after the sixth quarter of the program,[x] a very productive quarter for us. We developed and circulated to the Consortium members an IDL specification for a client-server scheduler architecture (see **Figure 31 CORBA IDL interface design**), and we experimented with this client-server interface using the original scheduler prototype as an instance of the schedule server and a new schedule-browsing UI as the client (see **Figure 32 Experimental client-server system**).





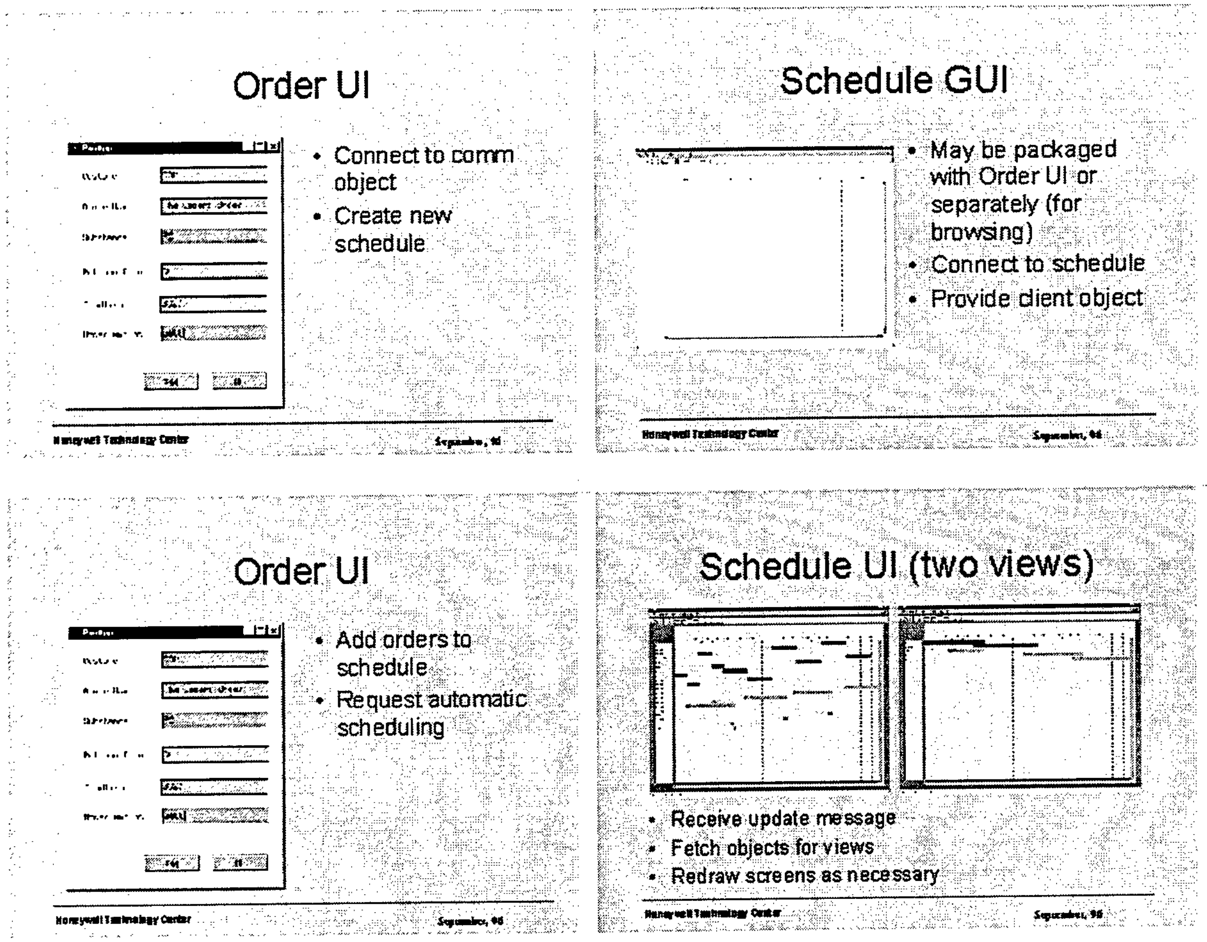
Interfaces

Interface Specifications	Key operations
Schedule Server	
Comm object	serves sets of schedule objects; supports creation and deletion of schedules
Schedule object	subscribe, unsubscribe, get view
View object	provides set of activities constituting a perspective on schedule
OrderServer	add_order and schedule method for schedules
Schedule Client	update

Honeywell Technology Center September, 96

Figure 31 CORBA IDL interface design

[x] At the fifth quarterly meeting we proposed this development to the Consortium but had little else to report, as our labor shortage recurred in this quarter.



Order UI

- Connect to comm object
- Create new schedule

Honeywell Technology Center September, 96

Schedule GUI

- May be packaged with Order UI or separately (for browsing)
- Connect to schedule
- Provide client object

Honeywell Technology Center September, 96

Order UI

- Add orders to schedule
- Request automatic scheduling

Honeywell Technology Center September, 96

Schedule UI (two views)

- Receive update message
- Fetch objects for views
- Redraw screens as necessary

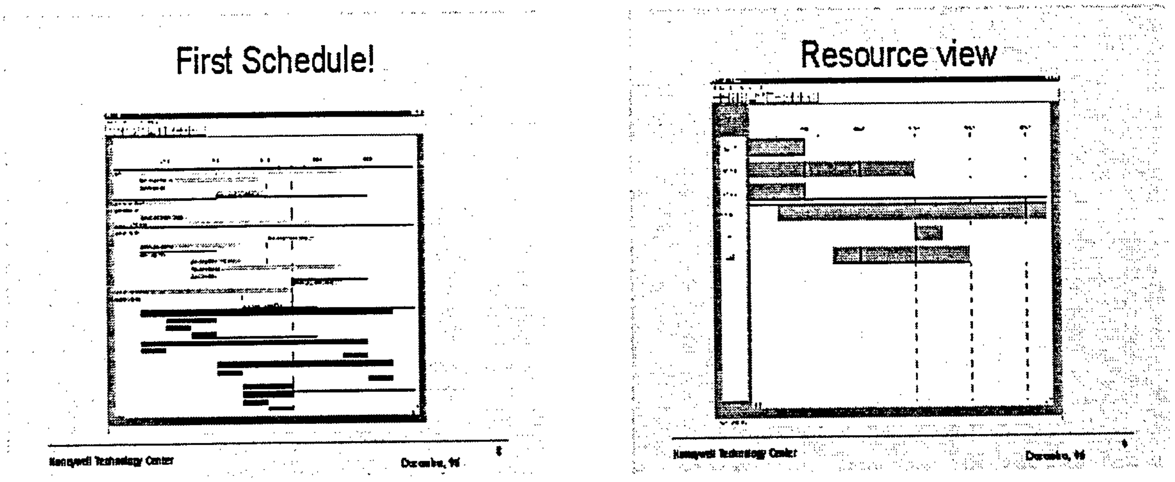
Honeywell Technology Center September, 96

Figure 32 Experimental client-server system

In the sixth quarter, we also completed our C++ reimplementation of the Common Activity and Task Model (CATM). At this point, we had in place a complete C++ implementation of the infrastructure for constraint-envelope scheduling.

At the review meeting for the seventh quarter, HTC unveiled the first full scheduling system in C++. This marked the completion of a batch manufacturing model based on the CATM developed earlier in the project. We demonstrated this system on a test problem and showed the results at the meeting (see **Figure 33 Results of scheduling test problem**).

Although at this point in the project, we had completed a full C++ implementation of the BMS, we were by now totally dissatisfied with C++ as an implementation language. Thus, this quarter marked the end of the C++ phase of our core implementation effort.



First Schedule!

Resource view

Figure 33 Results of scheduling test problem

7.4 Java implementation

The shift to Java began as a result of a training exercise[xi] by Mark Ringer, one of the engineers working on the BMS. As a way of learning the language, Mark translated the search engine into Java. The results were instructive: although the program ran somewhat slower, the slowdown was acceptable and the code was far simpler. Better yet, the code was more reliable as a result of being simpler.

Based on this initial experience, Java seemed a better programming language for our development purposes. However, that was not sufficient reason to switch. Two other considerations—technology transfer and ease of translation—combined to push us to the decision.

Our initial reason for switching from Common Lisp to C++ was a concern for technology transfer. Although the superior speed of execution of C++ has been widely reported, our experience did not bear this out. Rather, we found that the key to providing superior performance lay in being able to rapidly develop and deploy better algorithms and found this was far easier to do in Common Lisp. Note that we do not say that Common Lisp is *generally* better than C++, only better for the kinds of problems that interest us, the Automated Reasoning group at HTC. Those problems are likely to involve more complex and varying data types and more intensive computation than most programs, and certainly more than the systems programs for which C and C++ were designed.

So our decision to adopt C++ was driven by a need to be able to transfer our research systems to Honeywell divisions for productization. Accordingly, when we were considering Java as an alternative to C++, we consulted our divisions about the acceptability of Java and did not go ahead until they had deemed it acceptable.

Even though Java was acceptable to our divisions and easier for us to use, that was still insufficient reason for us to abandon C++. The final factor in our decision was

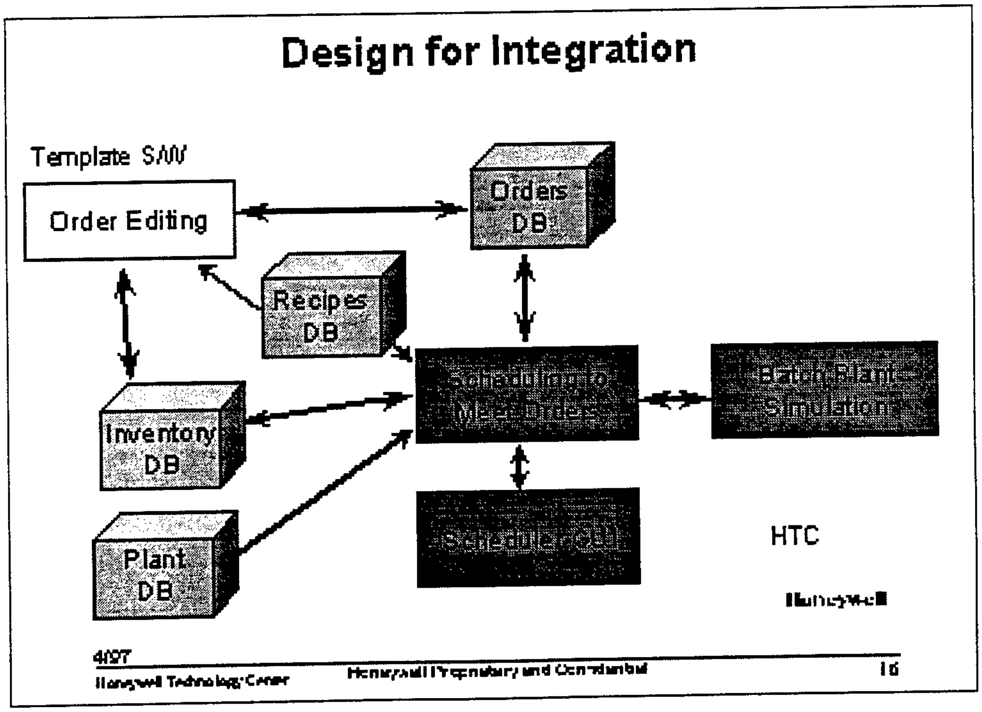
[xi] Funded by internal HTC resources rather than under the auspices of this program.

discovering how easy it would be to translate our existing C++ code into Java. We were pleased to find that, since the syntax was similar, and since Java has a simpler object model than C++, we could automate large parts of the translation process. Further, most of the hand editing needed to complete the transition consisted of removing from the code complications (particularly concerning memory management) that were necessary to C++ but that were handled automatically in Java. In fact, we were able to complete translation to Java of all the existing C++ code in only 2 man-months. The results bore out Ringer's experience with the search engine port: the resulting code was simpler and, as a result, contained fewer errors. We presented the Java translation at the meeting following the eighth quarter of the program, held in April 1997.

7.5 Enterprise Integration Demonstration

The eighth quarter also marked the beginning of our work with Template Software to develop the final version of the Enterprise Integration Demonstration (EID). Together, we decided that Template's Workflow Template was the appropriate vehicle for such an enterprise integration. For more details about WFT, see Section2. We also agreed that it would be best to communicate between components by means of shared databases.

The communication structure, together with the responsibilities of the two firms, is shown in **Figure 34 Architecture of Batch Plant Enterprise Integration Demonstration**. In this illustration, the shared databases are shown in blue, HTC's modules are shown in green, and Template's in gray. The shared databases were designed by HTC based on our expertise in batch manufacturing. They were implemented by Template Software using Microsoft Access™.


Design for Integration

4/97
Honeywell Technology Center
Honeywell Proprietary and Confidential
16

Figure 34 Architecture of Batch Plant Enterprise Integration Demonstration

The Batch Plant EID was completed in the ninth quarter of this program and demonstrated at the final quarterly meeting of the DASH'R Consortium. The demonstration was videotaped at HTC in Minneapolis, and a copy of the videotape is available.

7.6 Follow-on Work for Continuous Manufacturing

During the last six months of this project, we have been investigating extension of the batch manufacturing scheduler to cover continuous manufacturing (particularly petroleum refineries) as well as batch. This follow-on work has been "pulled" by marketing considerations and "pushed" by technological developments and by new technology transfer opportunities.

Honeywell traditionally has a very strong presence in continuous manufacturing such as petroleum refining, petrochemicals, and pulp and paper. Honeywell Industrial Automation and Control is the leading automation supplier to the hydrocarbon processing industry (HPI), including refining and petrochemical operations. Our automation systems are installed in 19 of the world's 20 largest integrated petroleum companies and in plants of 1 of the top 20 petrochemical producers. With their plants successfully controlled, Honeywell customers are interested in making them more profitable through more efficient scheduling of plant operations.

The key technological development that has pushed us to work on continuous manufacturing is a new understanding of the integration of discrete decision making, like that done in our previous schedulers, with continuous optimization, like that done by linear programming tools.

The key technology transfer development has been the acquisition of Honeywell Hi-Spec Solutions. The Hi-Spec Solutions group is a leading supplier of advanced computer applications, training, and services for the hydrocarbon processing and chemicals industries. Hi-Spec Solutions applications are engineered to ensure the production of high-quality products consistently, safely, and profitably. Hi-Spec Solutions has a staff of more than 600 technology experts with more than 6000 staff-years of experience. These technologists are primarily experienced process engineers with chemical engineering and process control training, but they also include a number of specialists in mathematics, software development, and information systems. We expect Hi-Spec to be the recipient of the technology transfer of the BMS. The fact that their primary focus is refining and petrochemicals, has helped direct us towards continuous manufacturing.

8. Development Tools

The development of the Honeywell BMS has been significantly influenced by the tools used. We have made extensive use of Computer-Aided Software Engineering (CASE) tools throughout the development process with varying success. We have used software for object-oriented design, both the mass-produced product, Rational Rose, and DoME™, developed at Honeywell Technology Center. Even with the design in place, no fewer than *four* programming languages were used in the course of this project: Common Lisp, SNAP, C++, and Java. Finally, in making this a distributed application, we had at our disposal two implementations of the CORBA standard: IBM's SOM/DSOM™ and Xerox PARC's ILU. We discuss our experiences with these tools in the following sections.

8.1 Design Tools

We used two design tools development of the BMS. Initially, we used Rational Rose to design our model of batch manufacturing enterprises; however, our experiences with Rational Rose were not satisfying, and we completed the design using our own product, the **Do**main **M**odeling **E**nvironment (DoME). Essentially, the difference between these tools lay in the extent to which we could specify and control the automatic translation of design specifications into actual program code. In this respect, DoME was far more satisfactory.

8.2 Rational Rose

Our initial design work[xii] in this project was done using Rational Rose. Prior to the development of UML, Rational Rose was an excellent editor for the Booch notation for object-oriented software design, "a software-engineering tool that allows users to graphically develop, verify, and document the analysis and design model of their software."[xiii]

Our experience with Rational Rose was not, however, satisfactory. We did not like its use of the Booch notation as much as the Coad-Yourdon format with which we were more familiar. Further, we found the generation of code from the diagrams insufficiently controllable.

We found the Booch notation overly restrictive and overly tied to the C++ programming language object model. For example, the Booch notation requires designers to indicate whether a given attribute of an object is "really there" in an object, or whether there is just a pointer to that object. This is a distinction critical in C++[xiv] but is a concept that does not make sense in other object-oriented languages, such as Java, Smalltalk, and Common Lisp. Not only is this distinction extremely language-specific, it does not seem appropriate to specify it as part of a high-level design process, since a decision about this in one place is intimately tied to the details of other decisions and one's memory

[xii] Over a period of approximately the first half year of the project.

[xiii] From the Rational Software Corporation web-site, November 1997.

[xiv] Not one of our favorite features of the language!

management strategy. We preferred the Coad-Yourdon modeling notation, which was more familiar to us through years of use in the DoME tool (see Section 8.2.1).

Our other reason for being dissatisfied with the Rose tool had to do with its automatic generation of program code. The version of the Rose tool we used took some (but not all) the diagrams drawn by users and automatically translated them into C++, which was problematic for two reasons. First, many of the diagrams generated as part of the Rose design process had no semantics; they did not yield either code or checkable requirements documents. Such diagrams present two problems: (1) since they are not aligned with program code, their content tends to stray as the program code is altered; (2) partly due to the first problem, they provide their reader with a false sense of knowledge about what the system designed will do. The other problem with Rose code generation was that we did not have sufficient control over the process. We are more familiar with the approach taken in our own DoME tool, with which one can actually program (or modify through programming) the code generation behavior of an editor. In this way, we could take design diagrams developed for the earlier, Common Lisp version of the batch scheduler and simply port them.

8.2.1 DoME

Honeywell's Domain Modeling Environment (DoME) tool set is an extensible collection of integrated model-editing, metamodeling, and analysis tools supporting a model-based development approach to system/software engineering. One of the key ideas behind DoME is that the design model is a primary representation of the corresponding piece of software. To that end, the design document supports automatic translation into program source code. We have made extensive use of this capability in building the Batch Manufacturing Scheduler.

8.2.1.1 Background

In model-based development, the model is the primary representation of the product. Automated transformations (e.g., code generators) convert the model into other forms needed for producing the product (e.g., source code). This implies, among other things, that engineers should treat a model as source code and rely on (automatic) transformation mechanisms to produce the object code. In return, this approach requires that the model specification tool(s) provide adequate expressive power to say all that need be said and execute the translation steps in a transparent, robust manner that is painless to the user.

In model-based development, the development process is partitioned into three components (see **Figure 35 Model-based design)** all of which our DoME tool supports:

1. Methodologists analyze modeling methods and build model-authoring tools to support the capture and management of domain-specific models.
2. Product developers describe the product or system being developed using formal modeling techniques and the model-authoring tools developed by methodologists.
3. Component/infrastructure developers use their knowledge of the target environment to (a) develop model analysis mechanisms that enhance model understanding, and (b)

transform models into software artifacts (source code, documentation, test cases), interface specifications, analysis algorithms, and generators (specialized back-ends).

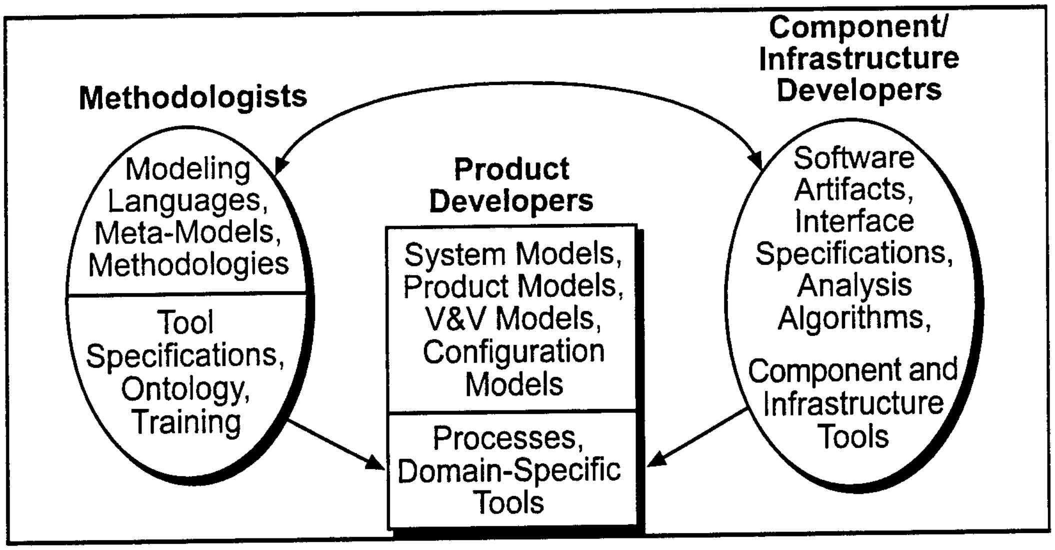

Methodologists
Modeling Languages, Meta-Models, Methodologies
Tool Specifications, Ontology, Training
Product Developers
System Models, Product Models, V&V Models, Configuration Models
Processes, Domain-Specific Tools
Component/ Infrastructure Developers
Software Artifacts, Interface Specifications, Analysis Algorithms,
Component and Infrastructure Tools

Figure 35 Model-based design

Since its beginning, the DoME project at HTC has sought to enhance the prototyping and production of graphical model-based development environments. DoME focuses on the second division of labor in the engineering process triad, providing powerful tools and support for product developers.

Model-based development requires model authoring tool(s) to provide rich, expressive power while employing a wide variety of abstraction techniques. DoME was developed for just this purpose. It supports a wide range of domain-specific graphical notations; more than 10 notations are included with DoME, and more than 40 have been used in the past. A screen shot of DoME being used to edit a Colbert object-oriented design diagram is shown in **Figure 36 DoME screen dump from design session.**

All of DoME's tools are based on a common foundation cultivated by HTC over the past several years. This foundation (GrapE) consists of a multilayered hierarchy of classes supporting both graphical model semantics and user interfacing. In addition to supporting multiple model-editing tools, GrapE provides a framework from which new, robust, domain-specific tools can be developed and displayed within a matter of a few hours to a few days. And DoME's thorough on-line help gives you convenient assistance for putting all of DoME's features to work for you. DoME's core features include:

- Enforcement of notation-specific syntactic and semantic constraints,
- Multiple hierarchical decomposition,
- Customizable model/diagram navigation,
- Alternative model views and overlays,
- User-defined, typed property annotations.

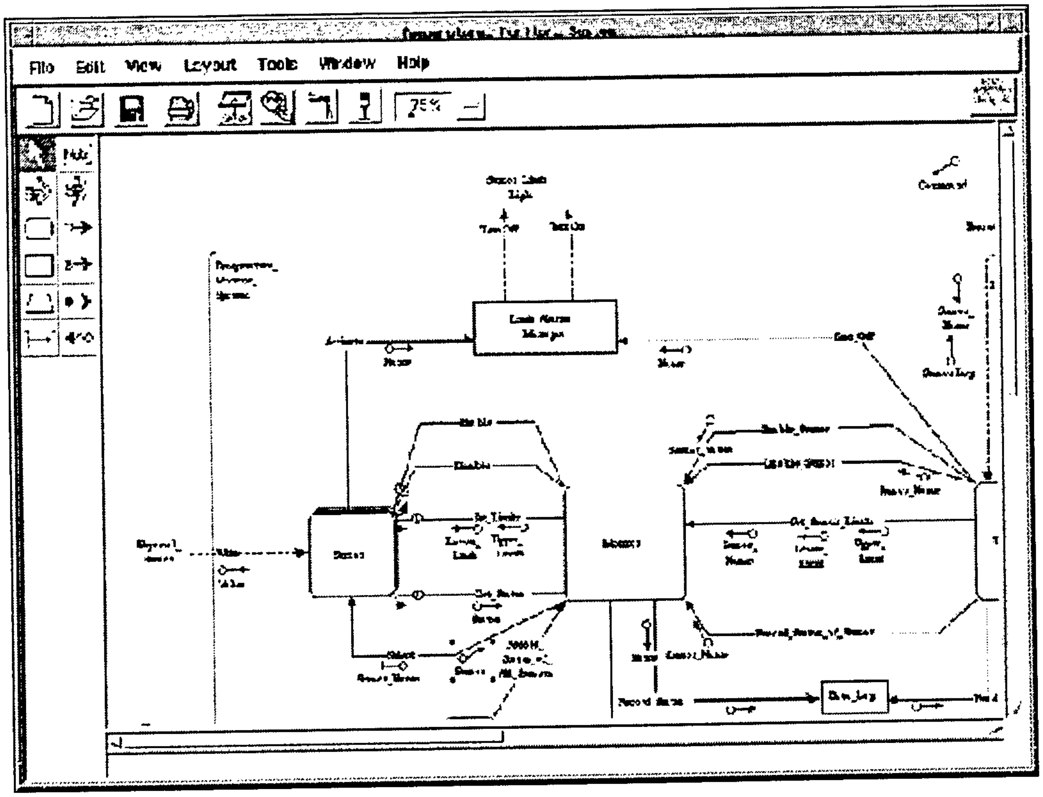


Figure 36 DoME screen dump from design session

Domain-specific syntax rules enforced. For example, our Petri Net model editor will not let you connect one transition, to another transition since Petri Nets are bipartite.

Change impacts automatically propagated. Changing a property of one visual object may affect the appearance of one or more related objects. For example, changing the name of a data flow in a parent Data Flow Diagram (DFD) will automatically change the names of all views of that flow in hierarchical subdiagrams.

DoME inherently supports reuse. Some DoME notations may contain a reuse repository called the "Shelf." Items placed on the Shelf can be reused with full traceability in subsequent diagrams.

Objects in hierarchical models can have multiple subdiagrams. In notations that provide this capability, the various subdiagrams, or "implementations," of parent objects are resolved through the use of configuration identifiers.

Nodes can contain things. Nodes can be adorned with other kinds of objects. For example, Petri Net places can contain marks (tokens) and Coad-Yourdon classes can contain lists of attributes and services.

In some notations, entire hierarchical subdiagrams can be contained and displayed from within a node or connector. You can directly manipulate these items in various ways (e.g., one common operation is to move them from one node to another).

Node size automatically determined. Node boundaries generally expand or shrink to fit the text inside. This is a design decision for each specific notation, which often results in cleaner, simpler interfaces.

Diagrams can interrelate. Diagram components can refer to other components or diagrams in separate models, through either hierarchy relationships or more general cross-reference relationships.

Because DoME's foundation is Smalltalk, it has a lot of flexibility and growth potential. New editors based on new visual grammars can be added, often in just a few hours. Most of the DoME tool set has been automatically generated using DoME Tool Specifications. When working with these specifications, you first enter a graphical, high-level specification of node and connector types, connection constraints, and additional syntax and semantics. The ProtoDoME tool for modeling tool developers can get you up and running with the prototype of a new tool in just minutes (see **Figure 37 ProtoDoME model**). And the Projector/Alter extension languages allow you to write new functions that are tightly integrated with DoME.

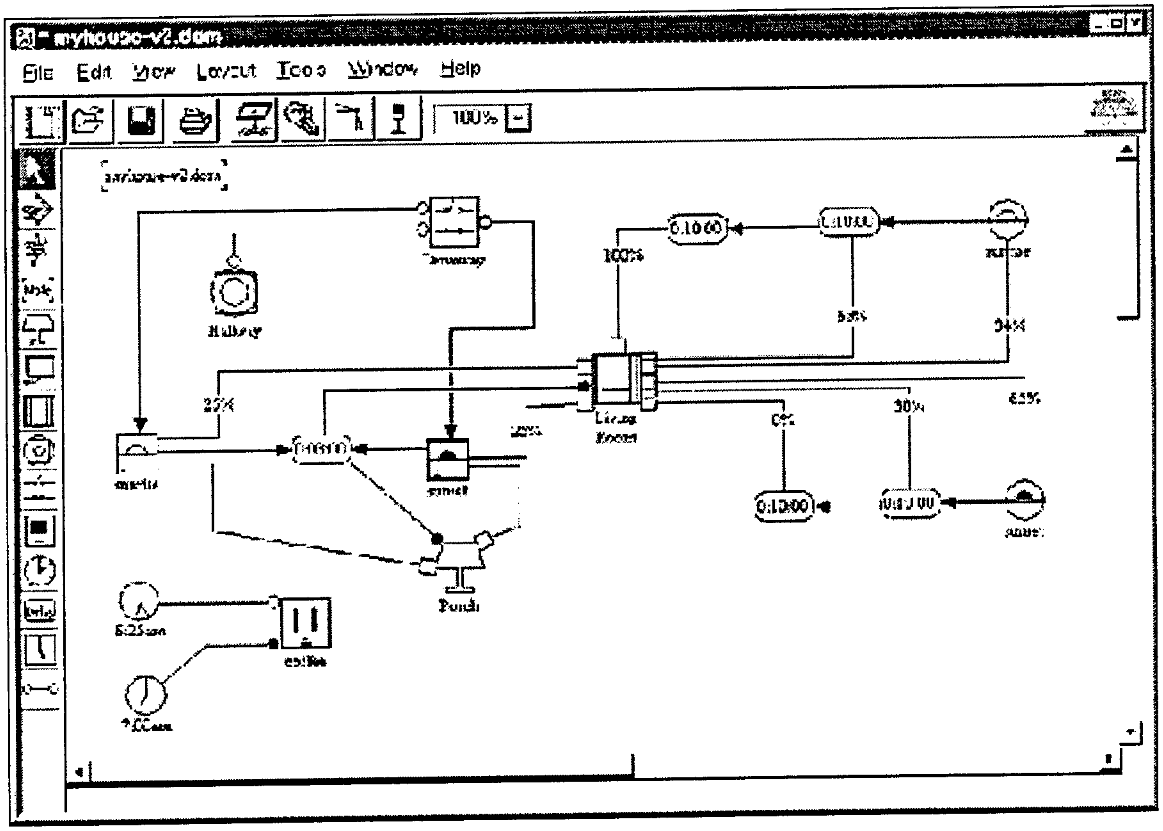


Figure 37 ProtoDoME model

Create New Graphical Languages. Using ProtoDoME—available with DoME 5.0—you can create new tools with DoME Tool Specifications. First, you begin to develop a new object model (class diagram) using the graphical DoME Tool Specification Language. You specify your visual notation by filling in properties on your object model. Your tool specification can include object classes, property definitions, relationship definitions, as well as connector types, dynamic object appearances, tool buttons, menus, annotations, semantic relationships, and other elements. Your graphical languages can also include textual, numeric, and symbolic annotations. A sample ProtoDoME diagram is shown in **Figure 37 ProtoDoME model**.

During development of your new notation, you can use ProtoDoME to run, view, and update your new model editor. Most changes to your DoME Tool Specification are immediately propagated to all relevant open models.

Generate Code and Documents from Models. DoME provides two artifact generation tools: Projector and Alter. These tools are built into and use the DoME infrastructure to assist in the extraction and manipulation of data represented in the foundation. Projector is a visual dataflow language; Alter is its functional textual cousin. Together they provide the functionality to write complex model transformations. Current uses include document, code, and test case generation, simulation and test execution, and model migration. (The end result of the transformation is really up to your imagination.)

We already support several common output formats. PostScript, Rich Text Format (RTF), Interleaf, and (Frame) Maker Interchange Format (MIF) are supported for documentation. Several software generators (specialized back-ends) have been written for the Coad-Yourdon tool that generate database schema code. In fact, you can write additional functions for any DoME tool.

Use Built-in Notations. In DoME 5.0 you have your choice of the following predefined notations: Coad-Yourdon OOA Colbert OOSD (three notations), Data Flow Diagram Document Outline DoME Tool Specification IDEF-0 and IDEF-1x diagrams Petri Net Projector Diagram (DoME visual programming system), State-Transition Diagram Colbert Object-Oriented Software Development (OOSD).

Ed Colbert, founder and president of Absolute Software Inc. and author of the Colbert OOSD method, has consulted with the DoME group at HTC during our implementation of his methodology.

On the surface, OOSD appears very much like many of the other OO methods, including Booch, Rumbaugh, and UML. OOSD differs in that the primary focus is on object interaction instead of object structure. OOSD also has fairly strict rules when crossing the boundaries between the Object Interaction, Behavior or State, and Class diagramming notations. These strict rules allow for a more maintainable and complete model that translates more readily to either Ada or C++ code. Currently, only one other application implements the Colbert OOSD methodology. Each has its strengths and weaknesses. Although the other product covers more of the notations, users have praised DoME for consistency maintenance, ease of use, and its powerful infrastructure.

Platforms. The platforms we currently support for DoME 5.0 are SunOS™ and Solaris™ (X-Windows™) and Windows™ NT and 95. DoME 5.0 is priced according to the number of concurrent users and has an annual subscription rate. Call for details.

8.2.1.2 Use in this project

The Automated Reasoning group at HTC has always made extensive use of the DoME tool. DoME was used in creation of the initial Common Lisp prototype of the BMS. Our initial design efforts for this project were made in Rational Rose (see Section 8.2 above), but we found this tool too limited for our liking. We switched back to DoME and developed the rest of our design using that tool. In particular, major chunks of our IDL interfaces and C++ code were designed and then automatically generated using DoME.

To support the design of the BMS, we developed three DoME code generators. These all took Coad-Yourdon OOA diagrams and translated them into program source code. We developed Alter (see previous section) filters that would translate Coad-Yourdon diagrams into: C++ header files (with function prototypes), C++ "skeletons" (boilerplate common to C++ programs; a simple labor-saving maneuver), and CORBA IDL. Ordinarily, we would have written a Coad-Yourdon-to-Java translator, but we generated most of our Java automatically from C++ source, instead of from the design document.

8.3 Programming Languages

In our work on the Honeywell batch manufacturing scheduler (BMS), we have used four programming languages, each with its own strengths and weaknesses. The initial prototyping effort was done (prior to the start of the OTRSD) using Common Lisp, and some additional work was done in Common Lisp early in the contract to quickly test features before reimplementation. The overall framework of the enterprise integration demonstration was provided by SNAP, particularly its Workflow Template. At the start of the program, core facilities were programmed in C++. We were looking for a lower-level language than SNAP to code up core capabilities. Unfortunately, C++ proved to be a poor choice. While C++ is widely used, and hence held out the promise of easy technology transfer, it did not provide facilities needed for programs like the BMS. Accordingly, in the last year of the program, we changed the implementation language for core facilities from C++ to the recently developed Java language. Java provided a better mix of high- and low-level programming and enabled us to finish the implementation successfully.

8.3.1 Common Lisp

The initial version of the Honeywell BMS was written in Common Lisp[xv] on Sun workstations. We have been very pleased with Common Lisp as a rapid-prototyping environment. The Common Lisp Object System (CLOS) provided the facilities needed for modeling the plant, recipes, resources, and so forth in a way that could easily be changed during the process of gathering more information about various kinds of batch manufacturing. The subtyping and method-specialization facilities of CLOS provided us with very effective mechanisms for code reuse. We have used the same core Common Lisp libraries to build scheduling systems for a wide variety of applications.

Although we have been happy with Common Lisp as a prototyping tool, it poses severe barriers to product deployment. The primary concerns are program size, user interfaces, and technology transfer. It is difficult to develop a reasonably sized executable system using Common Lisp. Lisp images are still very large. Another issue is that Lisp vendors seem to be behind C and C++ vendors (and far behind Java!) in providing multiple-platform user interfaces. Finally, conventional software organizations are unable to support and maintain Lisp code. This by itself effectively rules out any possibility of

[xv] Initially Lucid Common Lisp, later also Allegro Common Lisp.

providing a product in Lisp, since HTC's charter in Honeywell does *not* extend to final product development, deployment, and support; that responsibility rests with other Honeywell divisions.

8.3.2 SNAP

Our work on the BMS in this project assumed an enterprise integration framework for the demonstration system that was provided by Template Software through their Workflow Template. Template's SNAP product and its associated templates—System Management Template and Workflow Template—are described in Section 2.

8.3.3 C++

As mentioned earlier, our initial BMS prototype was developed in Common Lisp, but Common Lisp provided a poor choice for productization. SNAP provided the framework for enterprise integration, but we still needed a programming language for the core functionality, which is very compute-intensive. Our initial choice of programming language—prompted by concerns for technology transfer—was C++. Unfortunately, as we will explain below, C++ turned out to be a very poor fit for our requirements, and we redirected our efforts toward Java.

C++ is an object-oriented extension of the venerable systems-programming language C that has become the most popular object-oriented programming language now available. Fortunately for its success, but unfortunately for our purposes, two design philosophies guided its development: provide functionality that could be done immediately (at the time of its inception) and maintain compatibility with C.[xvi]

Unfortunately, these design philosophies led to C++ being a language without automatic memory management ("garbage collection") and with a very weak treatment of multiple inheritance. These limitations of C++ make it difficult to use in applications that must do a lot of memory allocation and deallocation and that have complex object models.

Unfortunately— for us, our scheduling systems have both of these characteristics: they are built on complex object models that represent the processes to be scheduled, and they make use of search algorithms that often allocate and free memory. Worse, the objects allocated and then thrown away are often complex objects with many pointers to other objects. This further complicates the task of memory management. Finally, our programs were initially written (and their algorithms designed) in the functional programming paradigm, in which functions are invoked and values are returned, and it is difficult to follow this paradigm in C++, since one must determine at what point a return value needs to be (can be) deallocated.

Another problem with C++ from our standpoint was that, despite being a standardized language, it was not easy to move C++ programs from one platform to another, and at HTC we use many different computing systems. For one thing, the C and C++ standards are notoriously weak where arithmetic is concerned. For example, it was difficult for us to get a 64-bit-long integer to use in our scheduling on both Windows NT and Unix.

[xvi] See Bjarne Stroustup, *The Design and Evolution of C++*, Addison-Wesley, 1994.

Another problem was implementations offering different subsets of the behavior the standard calls for. Finally, the standardization of libraries for the language has progressed more slowly than that of the language itself, causing there to be multiple different libraries for the same key operations, notably Microsoft's Foundation Classes and the Standard Template Library.

A final problem with C++ was the excessive complexity of writing software in this language. The excess complexity arises partly from the way object-oriented concepts have been grafted onto a very low-level systems programming language. This leaves the programmer with a great many details to track. There are the issues of memory-management discussed earlier, but there are also questions such as whether an object contains another object versus containing a reference to another object, the distinction between objects, pointers to objects, references to objects, and so on.

For all of these reasons, fairly late in the project, we became dissatisfied enough with C++ to want to abandon it. Fortunately, at this same time, Sun's Java programming language became widely available.

8.3.4 Java

Java provided a much better compromise between rapid prototyping and productization than did C++. It was superior for cross-platform portability and was also more genuinely object-oriented and hence far simpler. Further, Java, unlike object-oriented languages such as CLOS or Smalltalk, provided an acceptable path to technology transfer through its popularity.

Unlike C++, Java was designed from the beginning to be an object-oriented programming language.[xvii] As a result, the language is far simpler than C++. Unlike C++, it simply has objects, rather than objects and various kinds of references to those objects. Also unlike C++, it has no automatic type coercion feature to complicate the use of methods.

Again, because Java was designed from the bottom up to be object oriented, and did not have to maintain backward compatibility, it contains automatic memory management. The programmer is not responsible for tracking memory allocations and cleaning them up.

Finally, an important part of Java's *raison d'être* was achieving cross-platform portability. To that end, it has a very strong standard. Further, Java provides standard objects for user interfaces, filling a gap in C++.

Java uses C++ syntax, which has contributed to its widespread acceptance. The advantage this provided for us was that translation of our existing C++ code to Java could be done substantially automatically. We wrote a number of scripts (using **perl**) to translate C++ source to Java source automatically and thus were able to complete the transfer from C++ to Java rapidly.

[xvii] Although it still retains the distinction between primitive types and objects.

8.4 CORBA

One of the activities of this project was development of a version of Template Software's SNAP that would work with CORBA, a framework for distributed, object-oriented systems based on the notion of object request brokers. Template and IBM have successfully developed such a version (see their chapters of this report). We attempted to make use of CORBA in our BMS, and did experiment extensively with it, using both IBM's SOM/DSOM and Xerox PARC's Inter-language Unification (ILU) software. However, in the end, the CORBA framework was not sufficiently mature to be employed in a productizable manufacturing scheduling system.

An object request broker (ORB) is a middleware component that supports interobject communication in a distributed, multiplatform, multilanguage environment. OMG's Common Object Request Broker Architecture (CORBA) is an API and protocol specification for ORBs. CORBA defines an object model, an Interface Definition Language and interface repository, and services for invoking methods and managing object implementations. Bindings for C, C++, and Smalltalk are defined, as is a mapping to Microsoft's OLE/COM. Several vendors offer ORBs built to the CORBA specification. The common CORBA API permits application portability across ORBs; the common protocol permits interoperability among ORBs.

8.4.1 SOM/DSOM

The IBM chapter of the Consortium's document contains a detailed discussion of IBM's SOM/DSOM product, enhanced under this program. Unfortunately, although we experimented with this product fairly extensively during the project, it was not employed in the BMS. We would have liked to have access to a commercial, supported CORBA implementation, but we were never able to use SOM/DSOM. During this project, IBM's focus for SOM/DSOM shifted to supporting its own operating systems (e.g., on PC platforms, OS/2), rather than attempting to provide a solution for all operating systems. Accordingly, at no time during the project was there a fully functioning version of SOM/DSOM available that ran on Windows NT and worked with Microsoft's C++ compiler. Further, when we began to use Java, there was no SOM/DSOM protocol to work with Java,[xviii] and developing a suitable SOM/DSOM version was outside the scope of IBM's statement of work. Unfortunately, the constraints of our own commercialization plans precluded our developing a BMS that would only operate on IBM operating systems; Honeywell Industrial Automation and Control's standard platform for the future is Windows NT.

8.4.2 ILU

In early development work, we made use of Xerox PARC's Inter-Language Unification (ILU) framework to experiment with distributed, object-oriented systems. Xerox describes ILU as follows:

> The Inter-Language Unification system (ILU) is a multilanguage

[xviii] At the time of this writing, we do not know of any commercially-available Java ORB, able to interoperate with other ORBs.

> object interface system. The object interfaces provided by ILU hide implementation distinctions between different languages, between different address spaces, and between operating system types. ILU can be used to build multi-lingual object-oriented libraries ("class libraries") with well-specified language-independent interfaces. It can also be used to implement distributed systems. It can also be used to define and document interfaces between the modules of non-distributed programs. ILU interfaces can be specified in either the OMG's CORBA Interface Definition Language (OMG IDL) or ILU's Interface Specification Language (ISL).[xix]

For our purposes, ILU had two advantages: it was available for our C++ environment early in the project, and it had a defined Common Lisp interface. This permitted us to use ILU early in our project to examine how our system could employ CORBA. We conducted several experiments in which different components of the batch enterprise system (e.g., scheduler system and user interface, scheduler system and batch plant simulation) communicated through CORBA. However, ILU is not a commercially supported product, and not a component that would be acceptable to our divisions as part of a Honeywell product, so in the end the ILU work was scrapped. We had planned for that all along, intending to replace ILU with IBM's SOM/DSOM, but that was never possible (see preceding section).

8.5 Program Development Environments

For each of the three programming languages used on this program—Common Lisp, C++, and Java (SNAP is treated elsewhere at greater length)—we had to use at least one program development environment. In this section, we briefly discuss each of these environments.

8.5.1 Common Lisp

For our limited work on the previously existing prototype, we used Allegro Common Lisp on Sun workstations.

Our Lisp prototype uses Garnet[xx] for its graphical user interface. Garnet is a constraint-based user-interface construction toolkit developed at Carnegie-Mellon University and was funded by DARPA under its Human Computer Interaction (HCI) program.

[xix] From Xerox PARC's ILU web page: ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[xx] "Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces." Brad A. Myers, Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Ed Pervin, Andrew Mickish, and Philippe Marchal, *IEEE Computer*, Vol. 23, No. 11, November 1990.

Through its constraint-based structure, Garnet provided us with an ability to support rapid redesign of user interfaces. For example, when displaying objects corresponding to activities, we could simply define how they would appear in terms of a relationship between the duration of an activity, the width of the Gantt chart on which it was to appear, and the amount of time depicted in the Gantt chart. Little actual program code needed to be written, and these constrained objects could readily be reused. In turn, the ability to experiment with different user interfaces rapidly allowed us to home in on the needs of users by rapidly demonstrating working systems with very different GUIs.

8.5.2 C++

We used two C++ tools, depending on whether we were working on Sun™ UNIX™ systems or on Windows NT™. On the Suns, we used Sun's own SunPro C++ compiler, and on Windows NT, we used Microsoft's Visual C++.

With C++, we used the Rogue Wave™ libraries to provide general utilities and data structures. We also used XVT to provide cross-platform portable user interface objects.

8.5.3 Java

For our Java development, we used Symantec's Visual Café™ and then, when it became available, Visual Café Professional Development Environment™.

In our Java code, we have made substantial use of the ObjectSpace's Java Generic Library (JGL). JGL provides us standard data structures such as lists.

9. Commercialization

We plan to commercialize the scheduling technology developed under this TRP through Honeywell's Industrial Automation and Controls (IAC) division. At the time this project was conceived and proposed to DARPA, our original plans were to commercialize the resulting BMS through the Chemicals Industry business unit of IAC. However, several strategic changes in IAC made this infeasible. Current plans are to extend the BMS to cover continuous process industries (see Section 7.6) and take it to market through Honeywell IAC's Hi-Spec Solutions business unit.

9.1 Honeywell IAC

Industrial Automation and Control is a business unit within Honeywell's Industrial Control business. Industrial also includes the Sensing and Control business unit, based in Freeport, Illinois, and Honeywell-Measurex, a wholly owned subsidiary of Honeywell Inc., based in Cupertino, California. Honeywell Industrial Automation and Control's markets include the following industries: hydrocarbon processing; oil and gas exploration and transport; pulp and paper; chemicals; power generation; food, pharmaceuticals and other consumer goods; mining, metals and minerals; and semiconductors.

Honeywell, the world's leader in control technology, traces the origin of its industrial business to 1885, with the invention of a thermostat system to adjust the dampers of coal furnaces. It was one of the first control systems to employ electrical signals, and it helped establish an industry.

Honeywell formally entered the automation and controls business in 1934, when it purchased Philadelphia-based Brown Instrument Co. This business was strengthened in 1974 with the acquisition of General Electric's process computer operations in Phoenix.

The Industrial Automation and Control business was created in November 1989 with the consolidation of three operations: the Industrial Automation Systems Division in Phoenix, the Industrial Controls Division in Fort Washington, and the Industrial Services Center in York.

Honeywell pioneered the concept of distributed digital process control with the introduction of Total Distributed Control (TDC) 2000 in 1975.

The scope of process control was significantly expanded in the mid-1980s with the introduction of TDC 3000™. In 1992, the evolution of TDC 3000 continued with the introduction of TDC 3000X, which featured open systems access to plant information networks while maintaining a robust, secure pathway to the critical processes it controlled.

In conjunction with the announcement of TDC 3000X, Honeywell introduced **TotalPlant®** open solutions, which are best-value, integrated system solutions for defined industries consisting of products and services from Honeywell, supplemented by third parties, and built around an open Honeywell architecture. With **TotalPlant** solutions, a company can achieve the flexibility, product quality, productivity, and profitability needed to compete in today's marketplace.

In 1996, Honeywell introduced the TotalPlant Solution (TPS) system, the first industrial automation system designed to unify business and control information throughout a plant or mill. The new system takes full advantage of the power of Microsoft's Windows NT operating system. The benefits of the TPS system include higher overall productivity, and the ability to produce consistently high-quality products as economically as possible.

9.2 Original Commercialization Plans

Our original plans were to take the BMS to market in coordination with the Chemicals Industry vertical market unit of Honeywell IAC. We have had expressions of interest from the Chemicals vertical market organization, both in the United States and overseas (Europe and Australia); however, over the course of this project, the original commercialization plans had to be scrapped. Energy within the Chemicals vertical market has been refocused in directions other than scheduling, leaving no resources available to receive a technology transfer on this scale.

However, we have been in contact with Honeywell Hi-Spec Solutions (HSS), another component of Honeywell IAC, and are now working to take the BMS to market with that organization.

9.3 Honeywell Hi-Spec Solutions

Honeywell Hi-Spec Solutions is a leading supplier of advanced applications, training, and services for the hydrocarbon processing, chemicals, and pulp and paper industries. Formed by the merger of Honeywell Profimatics and SACDA operations, its best-in-class advanced applications and services are tailored for the process industries. They are engineered to ensure the production of high-quality products consistently, safely, and profitably.

The Hi-Spec Solutions misson statement is to improve the competitiveness of its customers by:

- Improving process profitability through automation,
- Improving process profitability through improved staff effectiveness,
- Reducing costs through technology and service.

HSS has grown rapidly over the years as a result of listening to its customers and successfully meeting their needs. HSS staff has grown to *more than 600* process experts in advanced applications and services for the process industries with *more than 6,000 staff-years* of experience. HSS is headquartered in Phoenix, Arizona, and holds offices in 16 other locations around the world.

9.4 Updated Commercialization Plans

HSS sees petroleum refining as the best initial market for HTC's scheduling technology. Accordingly, we are currently working with representatives of HSS and a major oil company to extend the BMS to be able to handle continuous processes such as petroleum refining. Our current expectation is for a pilot project in 1998, to be followed by productization in 1999.

9.5 Other Concurrent Commercialization Efforts

As this project has progressed, there has been growing interest in using the technology as the basis for other scheduling systems for Honeywell. In particular, we expect the core technology developed here to appear in scheduling systems sold by Honeywell's Commercial Aviation subsidiaries. In particular, we expect to provide scheduling functionality to the Honeywell Air Transport Systems Division AMOSS offering and the Honeywell Business and Commuter Aviation Systems RAMPS offering.

9.5.1 AMOSS

Air Transport Systems (ATS), headquartered in Phoenix, Arizona, along with Business and Commuter Aviation Systems (BCAS) in neighboring Glendale, make up Honeywell's commercial aviation business, each serving a segment of the market for commercial avionics. Air Transport Systems serves the market for large commercial airliners. It offers the widest range of commercial products and systems of any avionics supplier. ATS is also the world's premier systems integrator, with more experience managing major flight deck integration programs than any other company.

Both ATS and BCAS are making a major thrust in enterprise and operations integration for ground operations. ATS has established a new business unit within Air Transport Systems to provide data licenses for essential products and services to airline customers for use in simulator and flight training devices, for flight management systems (navigation database), to support the Boeing 777 aircraft (ground-based software tools), and for other maintenance and flight operations support systems for various aircraft types.

One such product is AMOSS, the Airline Maintenance and Operations Support System. In AMOSS, Honeywell's diagnostic technology, successfully applied on the Boeing 777, has been expanded to cover other airplanes and significantly improve the accuracy of diagnosis and reduce the time to maintain the fleet.

Using proven model-based diagnostic technology, AMOSS improves maintenance efficiency by:

- Reducing No-Fault-Found removals,
- Reducing repeat faults,
- Reducing maintenance-related delays and cancellations,
- Reducing maintenance backlog to support more aircraft.

Honeywell developed extensive capabilities in model-based diagnostic and troubleshooting systems as a result of experience on the 777 Central Maintenance Computer program. This experience, along with subsequent research and development, led to a cost-minimizing algorithm for troubleshooting an aircraft.

The system provides explicit direction for the optimal sequence of activities, as well as an option for the user to override the recommendation. All interactions and outcomes are recorded so that the algorithm actually learns from all of its "experience."

The diagnostics provide a mechanism by which testing and repair decisions are automatically analyzed for cost-effectiveness. Recognizing that activities are performed on a single aircraft by different crews at different facilities at different times, one important feature is that the system supports distributed fault isolation activities. The system implements record keeping and communication processes that allow this to occur with the same effectiveness as if they had been performed by a single crew at a single visit.

The overall approach provides a very accurate recommendation to the user that minimizes the cost of performing isolation and repair.

HTC is currently prototyping a maintenance scheduling system for AMOSS, and we anticipate that it will be added to the AMOSS package. The expected time of integration is toward the end of 1998.

9.5.2 RAMPS

BCAS designs, develops, manufactures, distributes, and services avionics systems and products for the worldwide business aircraft, regional airline, and helicopter marketplace. We provide these products and services to original equipment manufacturers (OEMs), regional airlines, and owners of aircraft in the after-market through dealers and completion centers.

BCAS is preparing the Ramp Asset Management and Productivity System (RAMPS), which will provide an integrated approach to asset management for baggage handling and ground operations. RAMPS will track ground assets, track resource utilization, and manage tugs. HTC's scheduling technology is expected to play a central role in this product.

10. Conclusions

Through DARPA's support, we have been able to apply state-of-the-art object-oriented software technology to the development of a scheduling system for dual-use batch manufacturing. We have taken research center software, partially supported by previous DARPA funding, and rebuilt it to be product-ready. We anticipate that this effort will lead to the offering of commercial scheduler systems through Honeywell's Industrial Control and Space and Aviation Control divisions.

HTC has also, in the course of this project, shown how Consortium-developed technology can be used as the basis for Enterprise Integration. Specifically, we have shown that Template Software's Workflow Template can be used as the communications backbone for an enterprise automation scheme uniting production planning, production scheduling, and plant operations

11. Screen Shots of BMS Prototype

11.1 Screen Shots of BMS Prototype

The screen shots in this appendix show a short scenario of interaction with the prototype Honeywell batch manufacturing scheduler. **Figure 38 Initial Screen** shows the initial screen layout. **Figure 39 Plant layout** shows the scheduler's plant layout display popped up, to examine the units in the plant and their connections. The next several figures show a scenario beginning with the entry of customer orders (**Figure 40 Order Entry**). The user schedules the batches necessary to meet these orders. Two views of the resulting schedule are given (**Figure 41 Initial schedule** and **Figure 42 Order view of initial schedule**), the first showing how the batches use units in the plant and the second showing how the user can track the completion of a set of batches for particular orders. We then show how to react to a plant upset. **Figure 43 The situation after BMS is informed one of unit has malfunctioned** shows what the scheduler screen looks like when the scheduler learns that a plant unit has malfunctioned. Finally, **Figure 44 Schedule after revision to compensate for malfunction** shows how the scheduler reacts to the malfunction, delaying some activities and reallocating others.

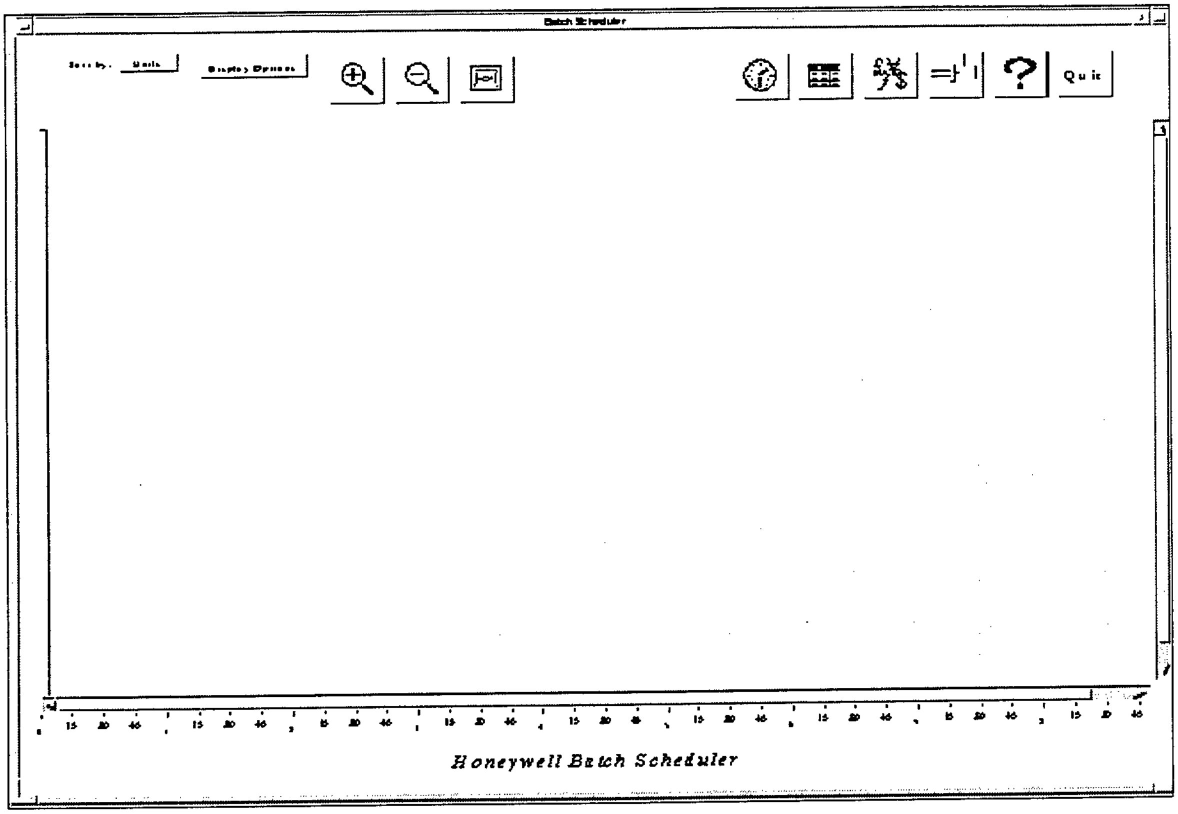


Figure 38 Initial Screen

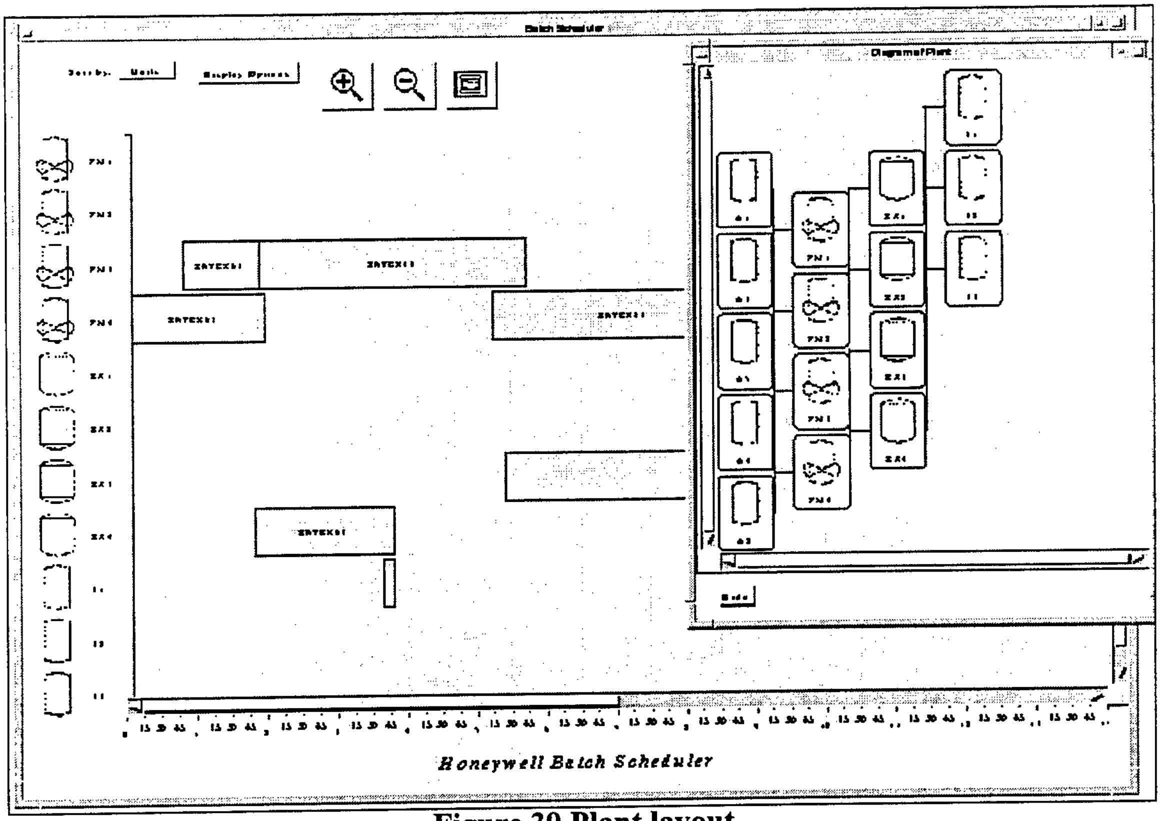
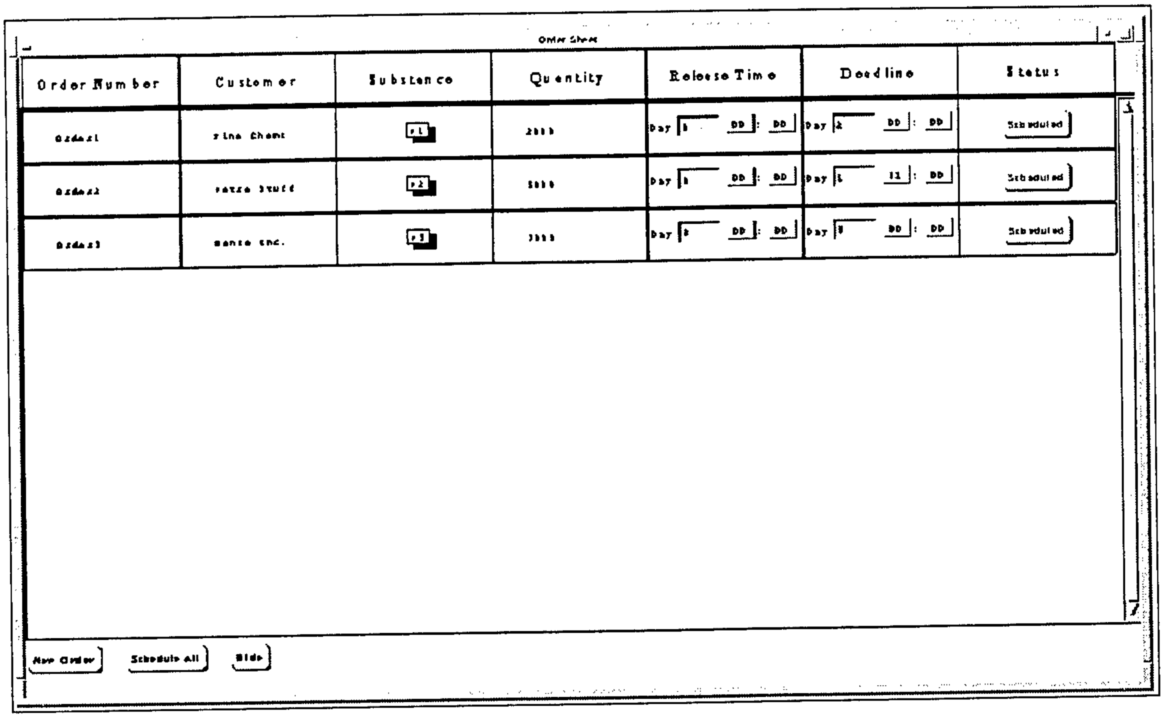


Figure 39 Plant layout



Order Sheet

Order Number	Customer	Substance	Quantity	Release Time	Deadline	Status
Order1	Fine Chemi	P1	[illegible]	Day 1 DD : DD	Day 2 DD : DD	Scheduled
Order2	[illegible] Stuff	P2	[illegible]	Day 1 DD : DD	Day 1 11 : DD	Scheduled
Order3	[illegible] Inc.	P3	[illegible]	Day 2 DD : DD	Day 3 DD : DD	Scheduled

Figure 40 Order Entry

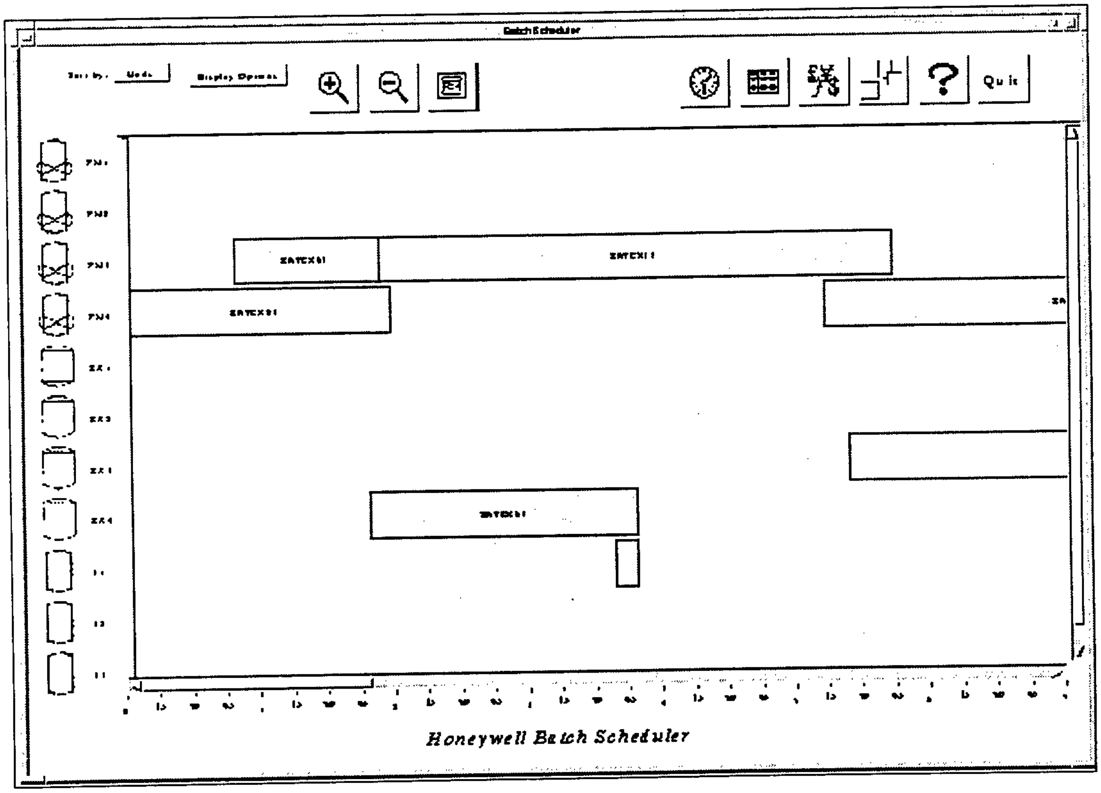


Figure 41 Initial schedule

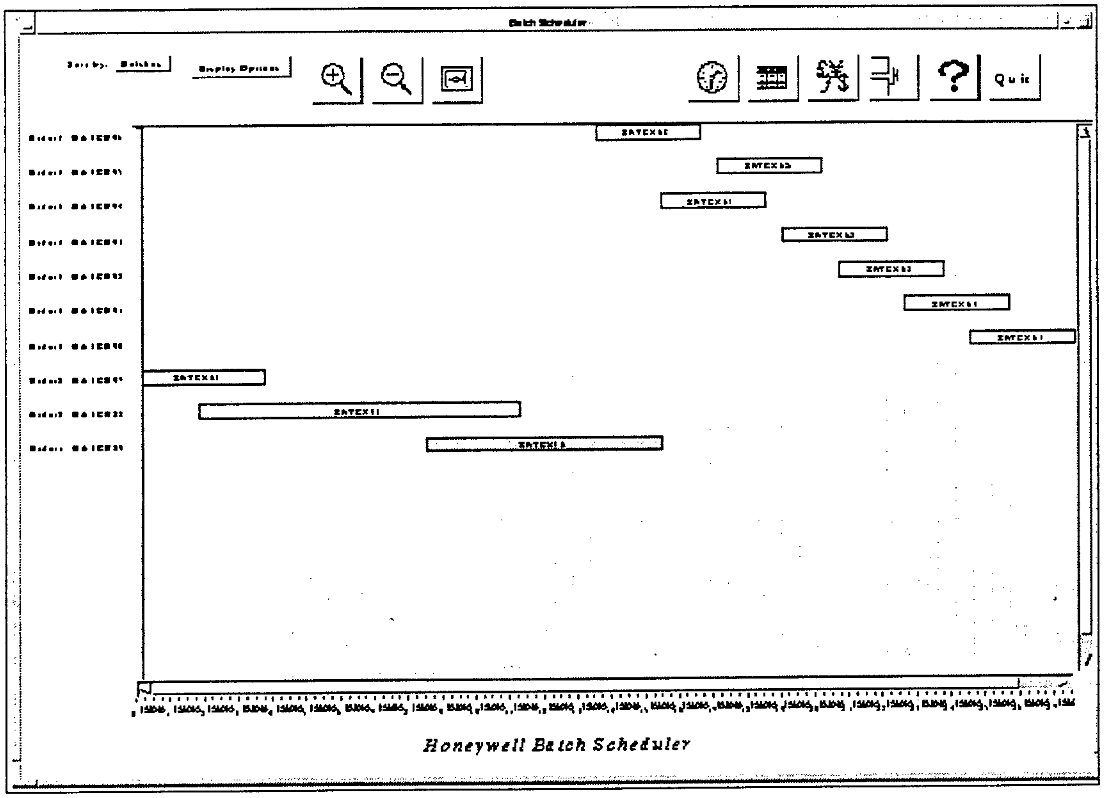
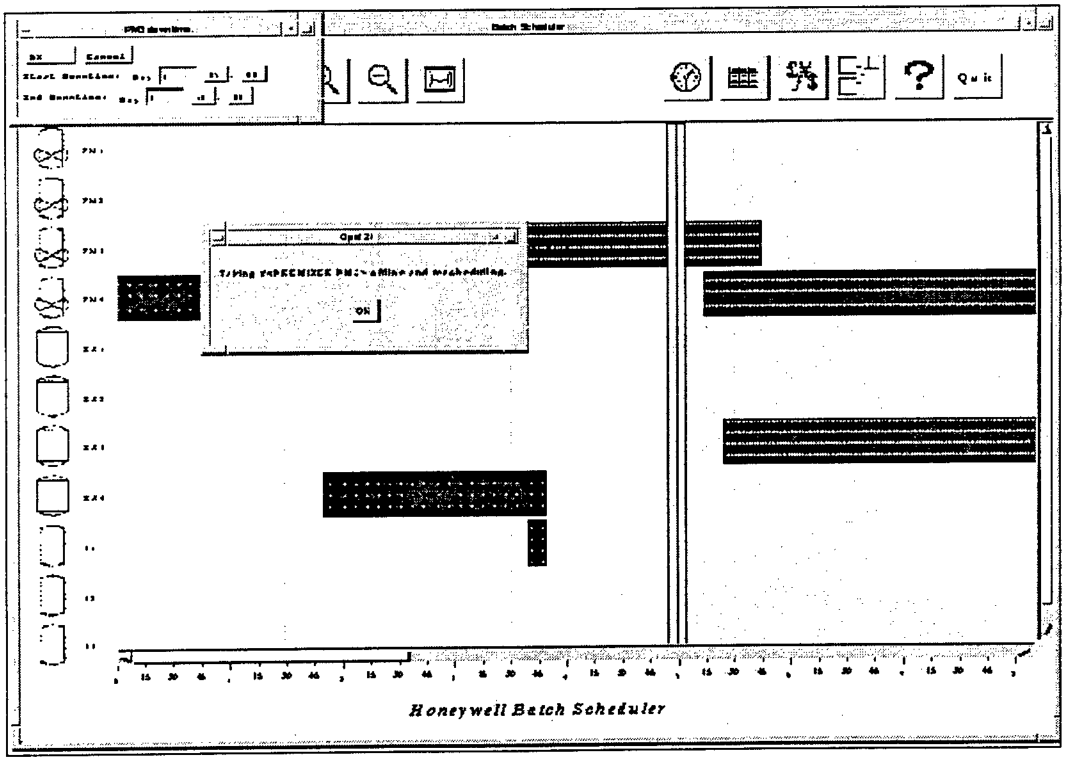


Figure 42 Order view of initial schedule



Taking PREMIXER PM[illegible] offline and rescheduling.

Figure 43 The situation after BMS is informed one of unit has malfunctioned

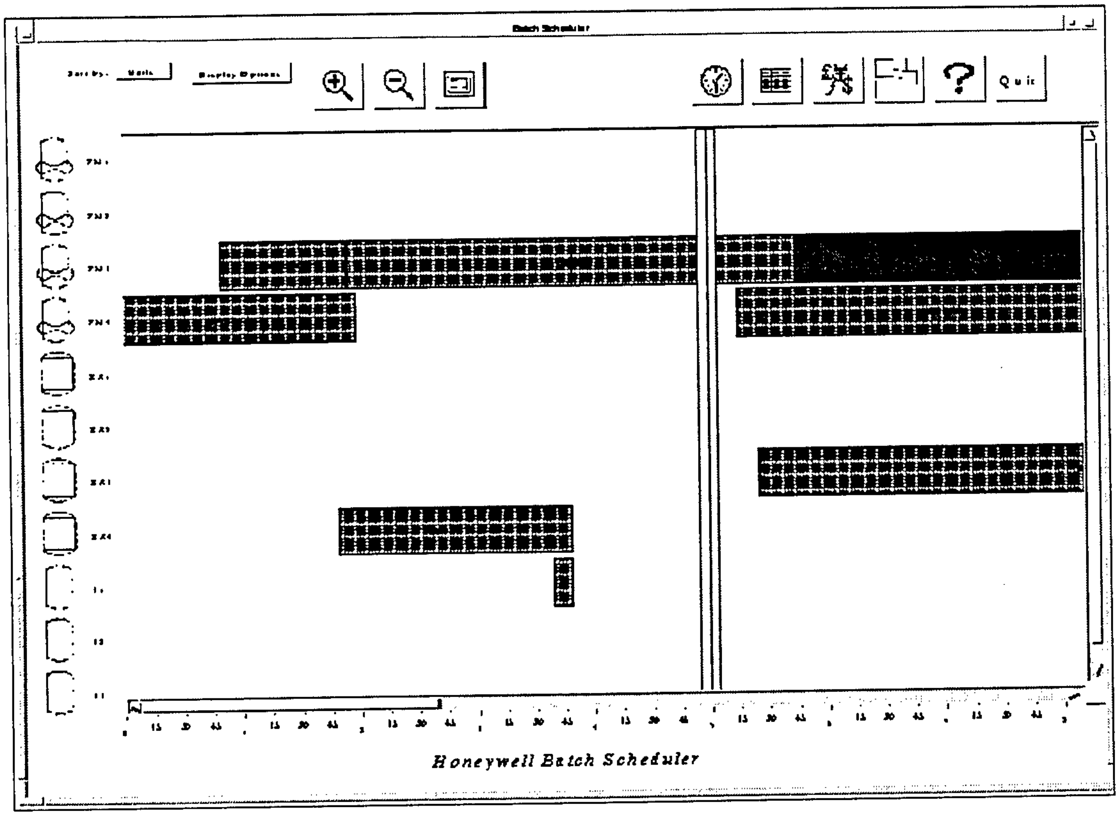


Figure 44 Schedule after revision to compensate for malfunction

DASH'R TRP
Final Report
IBM Corporation
April-95 through November-97

By: W. K. Centofonti

12. IBM

As part of the DARPA TRP, IBM plans on performing the following enhancements to SOM/DSOM technology. The enhancements fall into three categories: Object Services, ORB Interoperability and Core enhancements.

12.1 SOM / DSOM Development

As a consortium member of the DARPA TRP, IBM's tasks were focused on their deliverables which were elements of the Core Technology Base. These tasks were centered on a distributed object solution through enhancing and extending the SOMobjects technology. These enhancements fell into three major categories:

1. Object Services
2. ORB Interoperability
3. Core enhancements

The SOM/DSOM technology provides an object-structured protocol that allows applications to access and use objects, regardless of the programming language in which they were created and regardless of whether the object resides on host/client or client/server networks. For enterprise-wide distributed computing, this brings critical flexibility to object-oriented programming, object reuse and sharing. The Object Services portion of the deliverables provides application writers with an efficient and convenient way to use CORBA defined standard object API's and have access to services such as naming and externalization. The primary objective would be to provide a functionally sufficient set of object services to enable fully object-oriented application development around the CORBA implementation. Some of the services provided were implemented from scratch while others leveraged existing implementations (i.e. naming service). ORB interoperability permits objects residing on one CORBA-compliant implementation to interoperate with objects residing on another CORBA-compliant implementation. This is achieved through a common interoperability protocol between the two ORBs.

The core enhancements to the DSOM portion of SOMobjects deal mainly with making it a much more open architecture. The benefits of such openness are several: scalability, support for multiple protocols to co-exist (of which the CORBA interoperability protocol is one), customization with respect to storing DSOM repositories, leveraging target platform's architectural capabilities, and the ability to plug in different transaction/security implementations on top. The work was partitioned across the three primary areas of focus (Object Services, ORB Interoperability, and Core Enhancements).

The partition of work was further grouped into phases consisting of

Phase 1: Intitial design and implementation of object services, object service enhancements and core services.

Phase 2: Refinements of the phase 1 implementations and the design and implementation of ORB interoperability.

Phase 3: Refinements of ORB interoperability

Phase 4: Support for interoperability.

Each phase had related tasks to progress toward the final delivery. Tasks identified were Design, Integration, Test, Refinement, Packaging and Reporting. Testing of the enhancements and extensions to Object Services, ORB Interoperability and Core Enhancements was done on a multi-level basis. Functional verification testing was completed on each of the enhanced areas.

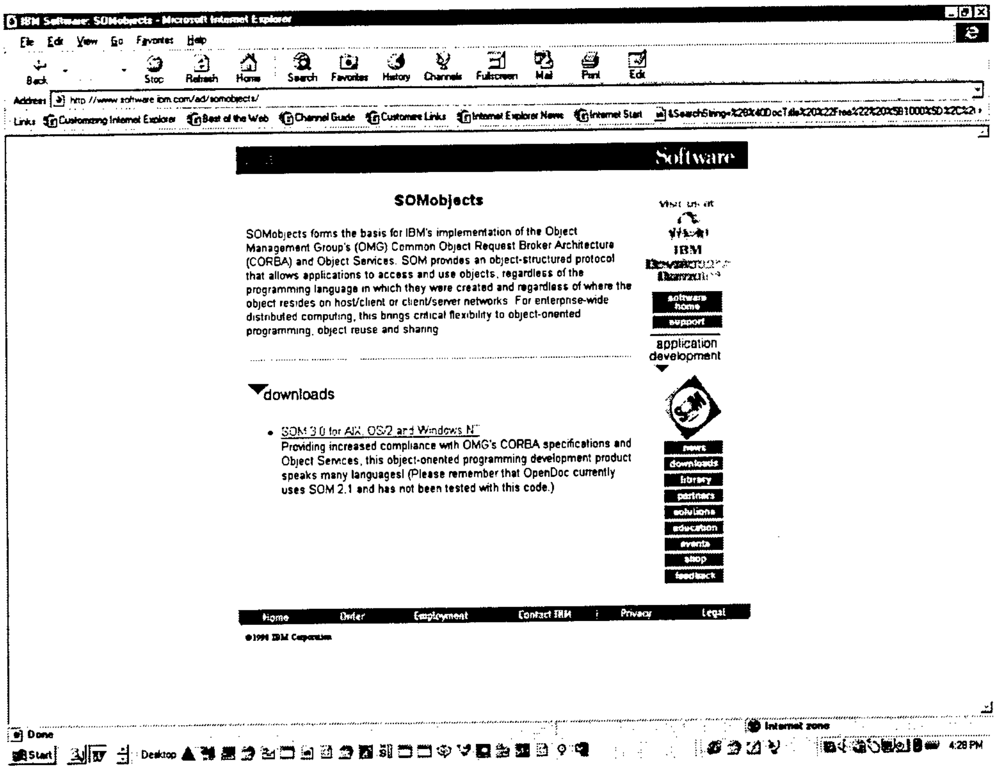
Early project drivers were made available to selected customers for a beta level of testing. Consortium members were members of this set of customers receiving the early project drivers. System Verification Test composed the final testing stage. This testing stage focused on the full project scope of function, performance and stability.

Special test arrangements were made with other ORB providers as a complement to the System Verification Test involving interoperability scenarios. Of particular focus were objects residing on one CORBA-compliant implementation interoperating with objects residing on another CORBA-compliant implementation. This demonstrated the flexibility of the project when using a standards-based object technology.

The results of this effort resulted in a commercially available product, SOMobjects R3.0 for AIX, OS/2, and Windows NT made accessible as of 12/31/96. The latest version of SOMobjects is available via the Internet at: http://www.software.ibm.com/ad/somobjects/

The SOMobjects R3.0 for AIX, OS/2 and Windows NT provides increased compliance with OMG's CORBA specifications and Object Services. It forms the basis for IBM's implementation of the Object ManagementGroup's (OMG) Common Object Request Broker Architecture (CORBA) and Object Services.

See **Figure 45 IBM SOMobjects WEB Page** and **Figure 46 IBM WEB Page For Selecting SOMobjects Downloads** below for the WEB pages that allow entry into the download process for IBM SOMobjects.


IBM Software: SOMobjects - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://www.software.ibm.com/ad/somobjects/
Links Customizing Internet Explorer Best of the Web Channel Guide Customer Links Internet Explorer News Internet Start
Software
Visit us at IBM
software home
support
application development
news
downloads
library
partners
solutions
education
events
shop
feedback
Home Order Employment Contact IBM Privacy Legal
©1998 IBM Corporation
Done
Internet zone
Start Desktop
4:28 PM

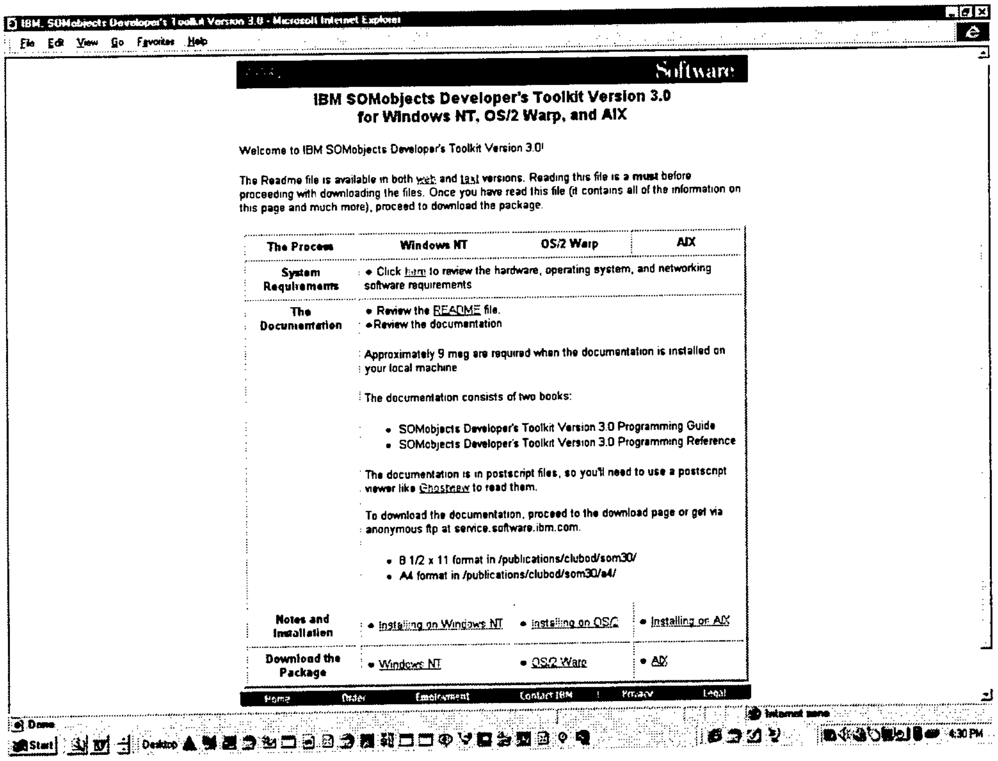
SOMobjects

SOMobjects forms the basis for IBM's implementation of the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) and Object Services. SOM provides an object-structured protocol that allows applications to access and use objects, regardless of the programming language in which they were created and regardless of where the object resides on host/client or client/server networks For enterprise-wide distributed computing, this brings critical flexibility to object-oriented programming, object reuse and sharing

downloads

- SOM 3.0 for AIX, OS/2 and Windows NT
  Providing increased compliance with OMG's CORBA specifications and Object Services, this object-oriented programming development product speaks many languages! (Please remember that OpenDoc currently uses SOM 2.1 and has not been tested with this code.)

Figure 45 IBM SOMobjects WEB Page


IBM. SOMobjects Developer's Toolkit Version 3.0 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Software
Home Order Employment Contact IBM Privacy Legal
Done
Internet zone
Start Desktop
4:30 PM

IBM SOMobjects Developer's Toolkit Version 3.0 for Windows NT, OS/2 Warp, and AIX

Welcome to IBM SOMobjects Developer's Toolkit Version 3.0!

The Readme file is available in both web and text versions. Reading this file is a must before proceeding with downloading the files. Once you have read this file (it contains all of the information on this page and much more), proceed to download the package.

The Process	Windows NT	OS/2 Warp	AIX
System Requirements	• Click here to review the hardware, operating system, and networking software requirements		
The Documentation	• Review the README file. • Review the documentation Approximately 9 meg are required when the documentation is installed on your local machine The documentation consists of two books: • SOMobjects Developer's Toolkit Version 3.0 Programming Guide • SOMobjects Developer's Toolkit Version 3.0 Programming Reference The documentation is in postscript files, so you'll need to use a postscript viewer like Ghostview to read them. To download the documentation, proceed to the download page or get via anonymous ftp at service.software.ibm.com. • 8 1/2 x 11 format in /publications/clubod/som30/ • A4 format in /publications/clubod/som30/a4/		
Notes and Installation	• Installing on Windows NT	• Installing on OS/2	• Installing on AIX
Download the Package	• Windows NT	• OS/2 Warp	• AIX

Figure 46 IBM WEB Page For Selecting SOMobjects Downloads

APPENDIX A: SOW

A-1. Introduction

The statement of work is organized by phase within consortium member plans. The effort increments are initially described and will be updated with finer granularity as the project progresses. The phases are shown in the Appendix B schedule.

The following sections present the SOW for each consortium member.

A-2. IBM Tasking for SOM & DSOM

As part of the DARPA TRP, IBM plans on performing the following enhancements to SOM/DSOM technology. The enhancements fall into three categories: Object Services, ORB Interoperability and Core enhancements.

The Object Services will permit applications to be written using CORBA defined standard object API's for such services as object naming, life cycle, and externalization. It is also planned to implement some extensions to these services that facilitate efficient and/or convenient use of these object services.

The ORB interoperability will permit objects residing on one CORBA-compliant implementation to interoperate with objects residing on another CORBA-compliant implementation. This is achieved through a common interoperability protocol between the two.

The third category of enhancements deals with the core enhancements in DSOM. They mainly deal with making it a much more open architecture. The benefits of such openness are several. To name a few: Scalability, Support for multiple communication protocols to co-exist (of which the CORBA interoperability protocol is one), customization with respect to storing DSOM repositories, leveraging target platform's architectural capabilities, and the ability to plug in different transaction/security implementations on top.

A.2.1 Work Effort Partitioning

A.2.1.1. Object Services

The primary objective is to provide a functionally sufficient set of object services to enable fully object-oriented application development around our CORBA implementation (i.e., SOM/DSOM). The following is the service set we plan to provide with SOM/DSOM. Some of these may be implemented from scratch while others may leverage existing implementations (for example, naming service).

* COSS-1 services (i.e., naming, and life cycle.)

* Enhancements to naming and life cycle services

* Security Service

A.2.1.2. ORB Interoperability: The ultimate objective is to implement the CORBA 2.0 interoperability specification and hence achieve interoperation between SOM objects running under IBM ORB and objects running under another (CORBA-compliant) ORB. The anticipated work products in this area are:

* Implementation of CORBA interoperability communication protocol as one of the supported communication protocol under DSOM.
* Implementation of interoperable object references
* Interoperation among different hardware platforms running SOM/DSOM

A.2.1.3. Core Enhancements: The main objective is to have an open DSOM that is flexible and scalable. The anticipated work products in this area are:

* New communication framework that allows multiple communication protocols to co-exist among client and server objects.
* Scalable architecture for repositories that the ORB relies on.
* Improvements in the design and implementation of CORBA architecture elements such as the object adapter, server object and CORBA pseudo objects.
* Architectural enhancements for incorporating security and transactions
* A marshaling engine for efficient marshaling of parameters

A.2.2 Phasing of Partitions:

Phase 1: Initial design and implementation of object services, object service enhancements and core services.

Phase 2: Refinements of the phase 1 implementations and the design and implementation of ORB interoperability.

Phase 3: Refinements of ORB interoperability

Phase 4: Support for Interoperability

A.2.3 Phase 1 Tasks

Task 1: Design, implementation, testing and integration of COSS services of Naming, and LifeCycle.

Task 2: Security-related architectural extensions: design, implementation and testing.

Task 3: Externalization service design, implementation and integration with other related services (e.g., LifeCycle)

Task 4: Extensions to base object services

- Extensions to Naming for properties and search capability
- Extensions to LifeCycle for creation of objects with specific properties
- Integration of services

Task 5: Communication Framework Reference Architecture: Design, implementation and testing with IPC based protocol. Documenting the reference architecture.

Task 6: Scalable Repositories: Design, implementation, and testing of new Object Registry, Implementation repository and modifications to interface repository.

Task 7: Improved design and implementation of CORBA architectural elements such as the Object Adapter for openness.

- New object reference design
- Re partitioning of functionality for better customizability (e.g., leveraging multi-processor architectures)

Task 8: Security: Architectural enhancements, design and implementation to allow the integration of a security service with ORB.

Task 9: A marshaling engine for marshaling parameters of remote calls.

Task 10: Documentation of DSOM (open) enhancements

Task 11: Object Services documentation

Task 12: Status reports and reviews

A.2.4 Phase 2 Tasks

Task 1: Refinements on DSOM enhancements and Object Services

Task 2: CORBA 2.0 Interoperability protocol: Design, implementation and testing of CORBA 2.0 mandatory protocol (based on TCP/IP).

Task 3: Implementation of Interoperable object references.

Task 4: Acquisition and test of another CORBA compliant ORB

Task 5: Test of interoperability between the two ORBs.

Task 6: Status reports and reviews

A.2.5 Phase 3 Tasks

Task 1: Port of SOMobjects to AIX.

Task 2: Port of SOMobjects to Windows NT.

Task 3: Reliability of Robustness enhancements to SOMobjects technology.

Task 4: Refinements of TCP/IP based CORBA Interoperability

Task 5: Status report/reviews

A.2.6 Phase 4 Tasks

Task 1: Support interoperability efforts.

A-3. Template Software Tasking for Core Technology Base

The Template Software objective is to create enhancements to its SNAP Product Family supporting consortium goals. The SNAP Product Family is object technology-based development environments for peer-to-peer distributed applications. Planned enhancements fall into three categories — SOM integration, C++ code generation, and visual tools development. SOM integration will provide a CORBA standards-based substrate to the SNAP Product Family for object interoperability and services. C++ code generation eliminates applications dependence on Template development tools for maintenance, in. step with TRP standards thrust and in the direction of interoperability at the development environment level. Incorporation of visual development tools will enhance SNAP Product Family's characteristics of rapid application and user-centered development.

A.3.1 Work Effort Partitioning

SOM integration extensions to the SNAP Product Family are independent from visual tool enhancement and define a natural work effort partitioning.

SOM Integration — the primary objective of SOM integration is to make the SNAP Product Family both a producer and consumer of reusable object services through SOM. To accomplish this, the External Application Software Component of SNAP will be extended to integrate SOM at the emitter framework level. This level of integration will make it possible for a SNAP developer to use externally created object components simply by writing method invocations in the SNAP language. Conversely, SNAP objects will be accessible to non-SNAP programmers writing in other SOM-compliant languages.

As ISX and Honeywell experience grows with this capability, other extensions regarding SOM may be made. Such extensions might include integration of selected SOM services and better leveraging of SNAP Product Family object services via SOM

C++ code generation — the SNAP Product Family Development Environments will be enhanced to allow generation of C++ class hierarchies

Visual Tools Enhancement — the SNAP Product Family Development Environments will be enhanced with new visual tools to simplify and speed up the development process. Anticipated extensions include:

- A new visual editor to graphically specify an application process object model.
- A language editor that can be invoked as needed by the various visual editors.
- Visual editors allowing end to end development, where end to end development includes business process analysis through system deployment
- Support for team development of multiprocess applications.

A.3.2 Phasing of Partitions

SOM integration and the Visual Tool Enhancement partitions each will have two phases. The phases will overlap with alternating milestones approximately at six month intervals to support the phasing of demonstration applications. Phase definition follows:

Phase 1: SOM Integration 1 – initial integration of the current SOM release with the current SNAP release.

Phase 2: Visual Tool Enhancement / C++ code generation 1 – first version of visual enhancements as part of the next planned release of the SNAP Product Family. Initial creation of internal infrastructure to allow C++ code generation

Phase 3: SOM Integration 2 – integration of the CORBA based SOM into the version of SNAP resulting from Phase 2 with other extensions deemed useful.

Phase 4: Visual Tool Enhancement / C++ code generation 2 – refinement of tools developed in Phase 2, visual tool extensions deemed useful, conclude C++ code generation capabilities and final demonstration of the development environments.

A.3.3 Phase 1 Tasks

Task 1: SOM Integration Design – define the design for integration of SOM emitter frameworks with the SNAP External Application component.

Task 2: SOM Integration – implement the Task 1 design. SNAP will be capable of acting in the role of client and server for SOM objects.

Task 3: Alpha Test – test SOM integration to demonstrate proper operation.

Task 4: Packaging – package the SNAP version including the SOM integration for distribution to the Demonstration projects. Packaging will include documentation on using the interface.

Task 5: Phase 2 Plan Refinement – refine and document scope and tasking for Phase 2 .

Task 6: Phase 3 Plan Refinement – refine and document scope and tasking for Phase 3.

Task 7: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.3.4 Phase 2 Tasks

Task 1: Prototype Visual Tool Concepts – create prototype of the visual tool interfaces.

Task 2: Tool Implementation - implement tools for object model specification, language editing, end to end development and multi-developer support

Task 3: C++ code generation - Initial design and development of internal infrastructure to allow C++ code generation

Task 4: Alpha Test – test tools in the context of integrated SNAP Product Family development environments.

Task 5: Packaging – package integrated SNAP Product Family development environments for distribution to the Demonstration projects. Packaging will include development environment documentation.

Task 6: Phase 4 Plan Refinement – refine and document scope and tasking for Phase 4.

Task 7: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate..

A.3.5 Phase 3 Tasks

Task 1: SOM Integration – integrate CORBA based version of SOM with SNAP.

Task 2: SOM-related Enhancements - based on Phase 1 & 2 feedback from the demonstration projects, make further SOM-related enhancements to SNAP improving SOM utility in the SNAP context.

Task 3: Alpha Test – test SOM integration to ensure proper operation.

Task 4: Packaging – package integrated SNAP Product Family development environments for distribution to the Demonstration projects.

Task 5: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.3.6 Phase 4 Tasks

Task 1: Prototype visual tool extensions - create prototypes of SNAP visual tool enhancements.

Task 2: Implement Extensions - implement visual tool enhancements.

Task 3: C++ code generation - finish C++ code generation capabilities

Task 4: Alpha Test – test integrated SNAP Product Family development environments.

Task 5: Packaging – package integrated SNAP Product Family development environments for distribution to the Demonstration projects.

Task 6: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A-4. ISX Tasking for Air Campaign Planner

ISX has two TRP major tasks: User Centered Software Development (UCSD) focal point for the consortium; and creation of an objectives-based Campaign Planning Tool (CPT) using the Core Technology Base (CTB) and the Strategic Planning Toolkit (SPT) infrastructure. The CPT will use the Air Campaign Planning Tool (ACPT) as a starting point for design and architecture. ACPT was developed by ISX and is a collection of cooperative components. This design is consistent with the development of an object-oriented software system. The components supply a complete collection of services and functions that enable the planning process. Viewers, assessment functions, analysis tools, scenario development and plan composition tools are all organized to support the systematic development of the campaign plan. The CPT architecture will be designed to take full advantage of the distributed object services and enable optional deployment in a multi-process, multi-processor, heterogeneous environment.

ISX will define and formalize UCSD concepts, methods, and processes for full use during this TRP. An initial document describing the process will be produced in the first phase with updates in subsequent phases.

The objective of the ISX effort is to define, design and prototype an objectives-based Campaign Planning Tool using ACPT as a starting point. This effort provides a specific application that will use the tools and development environment produced by the other team members, provide a requirements focus for the CTB and the main requirements focus for the Strategic Planning Toolkit (SPT). ISX will generalize the planning process embodied in ACPT and this TRP's CPT to create the initial infrastructure for a generic Strategic Planning Toolkit. An instantiation of the SPT will be a planning system with a planning process similar to ACPT's. Incremental prototypes of the CPT will be delivered at the completion of each phase. The first two phases' work will concentrate on requirements and infrastructure. The 3rd phase's prototype and the final system will incorporate greater degrees of functionality.

Continuation of the current tasking, with an update to the underlying PMPT database and the production of CDs that may be used to distribute the PMPT and its databases to users approved by the government sponsor.

Propose the additional development of a Medical Waiver System (MWS) to produce automating the medical waiver recommendation process. A government sponsor is interested in this work and it is an extension of the current PMPT development in the

Object Environment. This development could be commercialized, an objective of the TRP.

A.4.1 Work Effort Partitioning

The effort will focus on two items: the CPT application and the generic Strategic Planning Toolkit (SPT) infrastructure conducted in four six month phases.

A.4.2 Phasing of Partitions

Phase 1:

- Define User Centered Software Development process and capabilities for Consortium
- Provide expertise to the Consortium regarding the UCSD process
- Define SPT/ CPT requirements for CTB
- Define and develop the phase 1 CPT.

Phase 2:

- Update UCSD Process description
- Enhance CPT with new CTB features
- Define and develop the phase 2 SPT
- Increase CPT infrastructure and functionality

Phase 3

- Define and develop the initial SPT based upon CPT
- Enhance CPT with CTB upgrades, planned infrastructure and functions
- Implement initial interoperability capability

Phase 4:

- Update UCSD Process description
- Enhance SPT/CPT with final CTB

A.4.3 Phase 1 Tasks

Task 1: Define User Centered Software Development

Task 2: Define SPT/ CPT requirements for the CTB.

Task 3: Create phase 1 CPT definition.

Task 4: Design phase 1 / CPT

Task 5: Implement and test phase 1 / CPT

Task 6: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.4.4 Phase 2 Tasks

Task 1: Create phase 2 SPT/ CPT definition.

Task 2: Design phase 2 SPT/ CPT

Task 3: Implement and test phase 2 SPT/ CPT

Task 4: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.4.5 Phase 3 Tasks

Task 1: Create phase 3 SPT/ CPT definition.

Task 2: Design phase 3 SPT/ CPT

Task 3: Implement and test phase 3 SPT/ CPT

Task 4: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.4.6 Phase 4 Tasks

Task 1: Create phase 4 SPT/ CPT definition.

Task 2: Design phase 4 SPT/ CPT

Task 3: Implement and test phase 4 SPT/ CPT

Task 4: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.4.7 Phase 5 Tasks

Task 1: Create phase 4 MWS definition.

Task 2: Design phase 5 MWS

Task 3: Implement and test phase 5 MWS

Task 4: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A-5. Honeywell Tasking for Manufacturing Scheduler

The objective of the Honeywell effort is to create new manufacturing technology products and support consortium goals Honeywell will create a Batch Manufacturing Scheduling (BMS) application. BMS will be a mixed-initiative, on-line scheduling system for batch manufacturing. The scheduler will be based upon a batch scheduler prototype developed from DARPA-sponsored technologies.

The batch scheduler, for discontinuous manufacturing processes such as chemicals, composites, and pharmaceuticals, schedules production to meet orders. The schedules are produced from a plant model and recipes. The mixed-initiative scheduler will be both interactive and reactive to accurately reflect actual plan operations and will accommodate "what if" and financial analysis.

The foundation of the mixed-initiative scheduler is constraint-based scheduling technology by using the concept of a "constraint envelope" rather than explicit manipulation of individual scheduling events. Constraint-Envelope Scheduling closely relates to other constraint based systems. Constraint-Envelope Scheduling extends these traditional approaches with its support for temporal reasoning and through the adoption of critical principles such as "Open Scheduling" (dynamic user editing and inspection of partial schedules), "Least Commitment" scheduling (making scheduling decisions only when necessary), and explicit semantic representation of scheduling decisions in an active model. These techniques allow scheduling approaches to range from fully automated to completely directed through interaction.

Current state-of-the-art MRP II systems produce off-line, monolithic and inflexible scheduling outputs. These systems fall short of real-world requirements which must account for unpredictability and rapid change in operating parameters, interdependence with plant operations and control, and large bodies of human scheduling expertise which currently cannot be acquired in advance but must be captured at run-time through mixed-initiative user interaction.

Correctly handling continuous manufacturing requires the scheduler to be able to incorporate results of various numerical solvers, such as linear program solvers, etc. This addition is relevant to military operations planning, as well as manufacturing. E.g., the capability of interacting with special purpose numerical solvers is necessary to integrate path planning and fuel consumption concerns into existing air operations planners.

This work will also further the commercialization of the technology, one of the objectives of the TRP. The continuous process industries (e.g., oil refineries, paper and pulp, polyethylene) form the core of Honeywell's Industrial Automation business.

Extending the scheduler to continuous manufacturing will provide a much larger market and a more rapid application of the techniques developed under the TRP.

A.5.1 Work Effort Partitioning

The BMS application will be constructed as a single work product.

A.5.2 Phasing of Partitions

The manufacturing scheduling application will have four phases divided into six month increments. The phases are defined as follows:

Phase 1: BMS increment 1 (INC1)

Phase 2: BMS INC2

Phase 3: BMS INC3

Phase 4: BMS INC4

Phase 5: BMS INC5

A.5.3 Phase 1 Tasks

Task 1: BMS INC1 Definition – Current Honeywell scheduling technologies and prototypes will be reviewed to establish the product definition

Task 2: BMS INC1 Design

Task 3: BMS INC1 Implementation

Task 4: BMS INC1 Alpha Test.

Task 5: BMS INC2 Definition – the scope and tasking for Phase 2 will be defined and documented.

Task 6: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.5.4 Phase 2 Tasks

Task 1: BMS INC2 Definition – Current Honeywell scheduling technologies and prototypes will be reviewed to establish the product definition

Task 2: BMS INC2 Design

Task 3: BMS INC2 Implementation

Task 4: BMS INC2 Alpha Test.

Task 5: BMS INC3 Definition – the scope and tasking for Phase 3 will be defined and documented.

Task 6: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.5.5 Phase 3 Tasks

Task 1: BMS INC3 Definition – Current Honeywell scheduling technologies and prototypes will be reviewed to establish the product definition

Task 2: BMS INC3 Design

Task 3: BMS INC3 Implementation

Task 4: BMS INC3 Alpha Test.

Task 5: BMS INC4 Definition – the scope and tasking for Phase 4 will be defined and documented.

Task 6: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.5.6 Phase 4 Tasks

Task 1: BMS INC4 Definition – Current Honeywell scheduling technologies and prototypes will be reviewed to establish the product definition

Task 2: BMS INC4 Design

Task 3: BMS INC4 Implementation

Task 4: BMS INC4 Alpha Test.

Task 5: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A.5.7 Phase 5 Tasks

Task 1: BMS INC5 Definition – Current Honeywell scheduling technologies and prototypes will be reviewed to establish the product definition

Task 2: BMS INC5 Design

Task 3: BMS INC5 Implementation

Task 4: BMS INC5 Alpha Test.

Task 5: Reporting – support monthly status meetings and quarterly review meetings. Provide demonstrations at quarterly reviews as appropriate.

A-6. Interoperability Tasking

The primary objective of interoperability is to ensure that SOM/DSOM interoperates with other CORBA 2.0 compliant ORBs. IBM will perform this interoperability testing as detailed in A.2. The consortium will work ocoperatively with the other two TRP consortia to demonstrate ORB interoperability. ISX will play the lead role for this interaction.

A-7. Program Scheduler Management

The project will be directed by a Consortium Management Committee (CMC) according to the articles of collaboration. Day to day activities will be directed by individual company team leaders and overall coordination will be carried out by a program manager who will also be responsible for the following tasks:

Task 1:Conduct program kick-off review meeting

Task 2:Conduct monthly project status meetings

Task 3:Conduct quarterly review meeting(s)

Task 4:Submit progress reports.

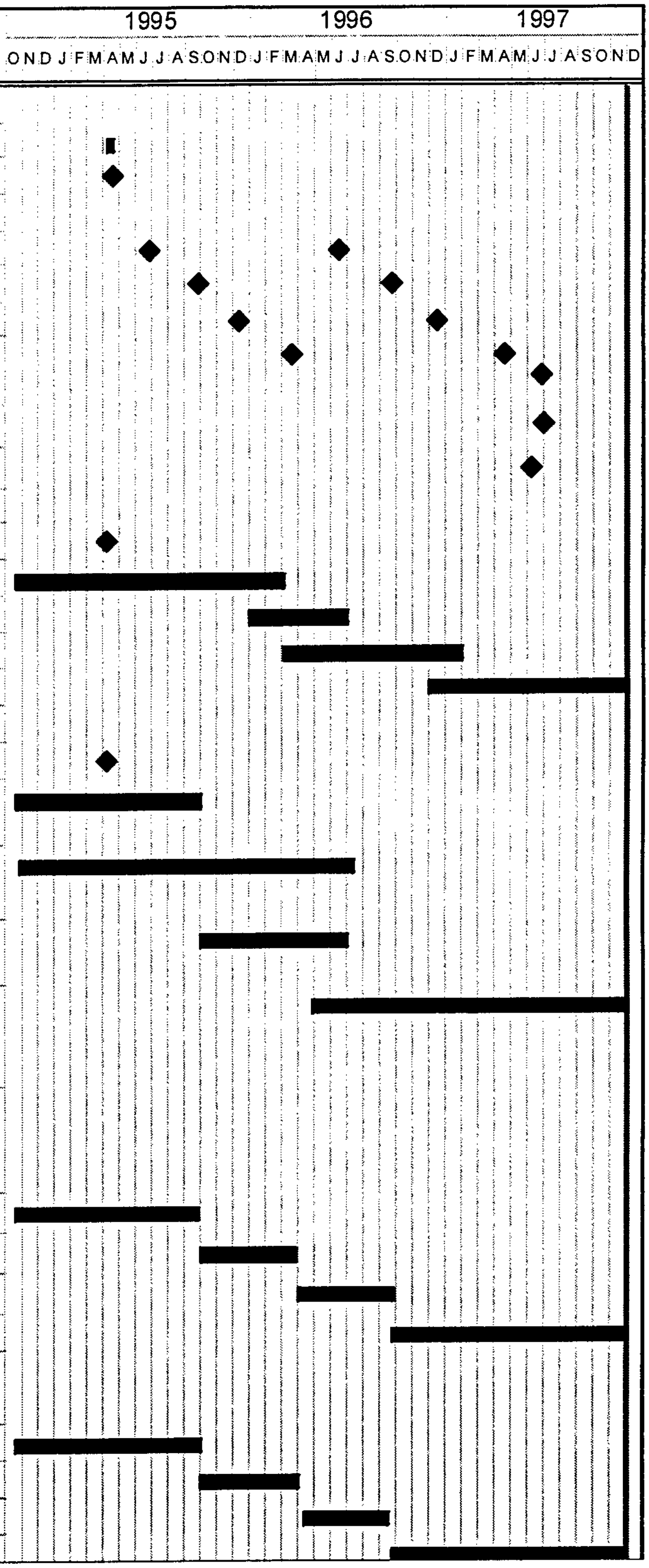
Task 5:Consolidate and coordinate initial project work plan..

Task 6:Update and coordinate project work plan.

Task 7:Conduct end-of-project review meeting and complete final report.

APPENDIX B: Schedule

Obj Tech (Rapid SW Dev) Sked.R3	Task Resp
A. Program Management	PM
Project Startup/ Initialization	CON
Project Status Review - Kickoff	CON
CMC Quarterly / Tech Status Rep	CON
Honeywell - 6/28/95, 6/19/96	
IBM - 9/27/95, 9/27/96	
ISX - 12/13/95, 12/11/96	
Template 3/20/96, 4/22/97, 6/30/97	
PSR/ TSR - End-of-project	CON
Project Close Out	CON
B. SOM	IBM
0. SOM 2.1 distribution to TSI	
1. Phase 1 Interoperable SOM	
2. Phase 2 Interop. SOM Final	
3. Phase 3 SOM Exts. / Port	
4. Phase 4 Interop. Supt.	
C. Core Technology Base	TSI
0. SNAP Distro. to Developers	
1. Phase 1 D/ SOM - SNAP Initial Integration	
2. Phase 2 Visual Programming Tools / Initial C++ Generation	
3. Phase 3 D/ SOM - SNAP Final Integration	
4. Phase 4 Visual Programming Tools Ext / Final C++ Generation (SNAP 8.0)	
D. Campaign Planning Tool (CPT) & Strategic Planning Toolkit (SPT) Define / Develop	ISX
1. Phase 1 C/SPT INC1	
2. Phase 2 C/SPT INC2	
3. Phase 3 C/SPT INC3	
4. Phase 4 C/SPT INC4	
E. Batch Manufacturing Scheduling (BMS) App Define and Develop	HW
1. Phase 1BMS INC1	
2. Phase 2 BMS INC2	
3. Phase 3 BMS INC3	
4. Phase 4 BMS INC4	



APPENDIX C: Honeywell Glossary and Acronyms

BMS (Batch Manufacturing Scheduler): We use this term to refer to our *automated scheduling system*, as opposed to a human being who is responsible for scheduling activities in a batch manufacturing plant.

CATM (Common Activity and Task Model): Set of object classes representing activities, resources, and resource requirements common to schedulers written at HTC.

CORBA (The Common Object Request Broker Architecture): A framework for distributed, object-oriented systems.

DCS (Distributed Control System)

EID (Enterprise Integration Demonstration): The overall enterprise management system, comprising production planning, BMS, and plant control system simulation.

GUI (Graphical User Interface)

IDL (Interface Definition Language)

ILU (Inter-Language Unification): A framework for combining object-oriented programs in different languages developed at Xerox PARC; it provides behavior that is a superset of CORBA.

ISA (Instrument Society of America)

JIT (Just in time): A style of manufacturing; in a JIT enterprise, products are made only in response to customer orders. This distinguishes JIT manufacturing from more conventional inventory-maintenance manufacturing, in which a fairly steady inventory of products is maintained in order that they be available when customer orders arrive.

MRP, MRP-II (Materials Requirement Planning)

OOA (Object Oriented Analysis): In particular, diagrams in the Coad-Yourdon design method are referred to as OOA diagrams.

ORB (Object Request Broker): Key component of a CORBA-based distributed system.

SP88: Batch manufacturing standard of ISA.

TMM (Time Map Manager): HTC's first temporal reasoning system, developed under a DARPA contract for the DARPA/Rome Laboratory Planning Initiative.

WFT: SNAP's Workflow Template, an object-oriented software template containing a standard model and active components for managing workflow in an enterprise.

☆U.S. GOVERNMENT PRINTING OFFICE. 2000-510-079-81252

DISTRIBUTION LIST

addresses	number of copies
NANCY A. ROBERTS AFRL/IFTD 525 BROOKS ROAD ROME, NY 13441-4505	5
LARRY SENTMAN TEMPLATE SOFTWARE, INC. 13100 WORLDGATE DRIVE SUITE 340 HERNDON, VA 22070	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/MLME 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739	1

AFRL/HESC-TDC 1
2698 G STREET, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604

ATTN: SMDC IM PL 1
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

TECHNICAL LIBRARY D0274(PL-TS) 1
SPAWARSYSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

ATTN: D'BORAH HART 1
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

ATTN: KAROLA M. YOURISON 1
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

AFRL/IFT 1
525 BROOKS ROAD
ROME, NY 13441-4505

AFRL/IFTM 1
525 BROOKS ROAD
ROME, NY 13441-4505

CENTRIC ENGINEERING SYSTEM, INC. 1
624 EAST EVELYN AVENUE
SUNNYVALE, CA 94086-6488

FLUENT INCORPORATED 1
500 DAVIS STREET, SUITE 600
EVANSTON, IL 60201

THE MACNEAL-SCHWENDLER CORPORATION 1
815 COLORADO BOULEVARD
LOS ANGELES, CA 90041-1777

MOLECULAR SIMULATIONS, INC. 1
9865 SCRANTON ROAD
SAN DIEGO, CA 92121-3752

CENTRIC ENGINEERING SYSTEM, INC.
624 EAST EVELYN AVENUE
SUNNYVALE, CA 94086-6438

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.